Zaawansowane metody programowania obiektowego (1-2)

dr inż. Dariusz Pierzchała

dariusz.pierzchala@gmail.com

Wykorzystano: materiały pomocnicze do książki Iana Sommerville'a „Inżynieria oprogramowania", wykłady prof. Kazimierza Subiety, szeroko dostępną literaturę…

Program zajęć

- Wprowadzenie w projektowanie i programowanie obiektowe
  - Metody obiektowe projektowania oprogramowania
  - Elementy notacji UML
- Zaawansowane techniki programowania obiektowego w językach obiektowo-zorientowanych
- Wzorce projektowo-programowe
- Programowanie aplikacji internetowych

Zaliczenie przedmiotu

- Zaliczenie przedmiotu - na podstawie: egzaminu oraz zaliczenia zajęć laboratoryjnych.
- **Egzamin** w formie: **egzaminu pisemnego - test z teorii i zadania z programowania**.
  - Podczas egzaminu pisemnego nie można korzystać z żadnych materiałów.
  - **Warunek dopuszczenia** do egzaminu: uzyskanie **zaliczenia z zajęć laboratoryjnych**.
  - Stosowana jest następująca reguła zaliczeń:
    - ocena dst: minimum 50 % możliwych do uzyskania punktów,
    - ocena dst+: minimum 60 % możliwych do uzyskania punktów,
    - ocena db: minimum 70 % możliwych do uzyskania punktów,
    - ocena db+: minimum 80 % możliwych do uzyskania punktów,
    - ocena bdb: minimum 90 % możliwych do uzyskania punktów.

Zaliczenie przedmiotu

- Wszystkie konsultacje odbywają się ul. Domagalskiego 7a
- Terminy:
  - 2016-10-14, 8.50-9.35
  - 2016-10-28, 8.50-9.35
  - 2016-11-18, 8.50-9.35
  - 2016-12-02, 8.50-9.35
  - 2016-12-16, 8.50-9.35
  - 2017-01-13, 8.50-9.35
  - 2017-01-20, 8.50-9.35
  - 2017-01-27, 9.45-10.30

Literatura podstawowa

- Metody obiektowe w teorii i w praktyce – Ian Graham, WNT, 2004
- Podstawy metod obiektowych – J. Martin, J.J.Odell, 1997
- UML – przewodnik użytkownika – Grady Booch, James Rumbaugh, Ivar Jacobson, WNT, 2001
- Język C++ – Bjarne Stroustrup, 1997
- Symfonia C++ – Jerzy Grębosz, 1999
- Bruce Eckel. Thinking in C++. Edycja polska. Helion. 83-7197-709-3
- Ian Sommerville, „Inżynieria oprogramowania”, WNT, 2003

Startujemy!

Trochę historii …

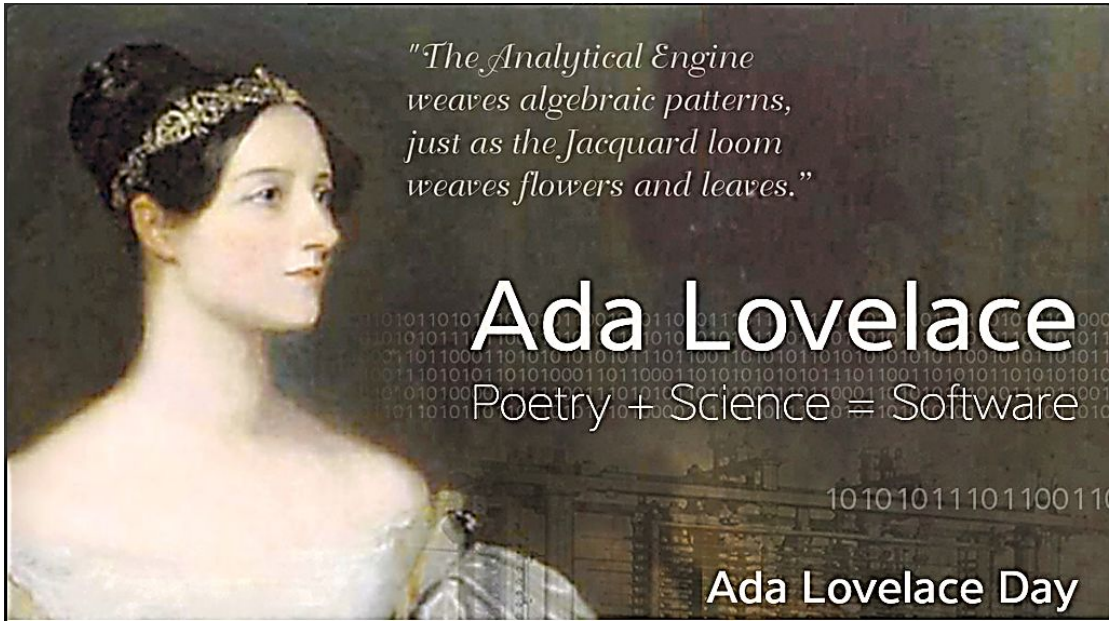
- Babilon (1790 p.n.e.) – tablice m.in. o zawartości matematycznej, astronomicznej:

Znana tablica z formułami twierdzenia Pitagorasa $a^2+b^2=c^2$ *(podobno jest tam błąd – kto odnajdzie?)*

- Bagdad (780-850) – matematyk Mohamm… zapisał w podręczniku pierwsze algorytmy dla systemu 10-ego z zerem;
- 1623 – Wilhelm Schickard z Tubingen skonstruował sumator liczb do 6 cyfr;
- 1812 – Charles P. Babbage opracował i zbudował mechaniczną "maszynę różnicową", wykonującą skomplikowane działania metodą powtarzania kombinacji elementarnych operacji;

Trochę historii …

- 1840 – Augusta Ada, córka lorda Byrona, od 19-tego roku życia po ślubie Lovelace – opublikowała pracę na temat dorobku Babbage'a.
  W swoich notatkach zawarła przemyślenia dotyczące przewagi systemu dwójkowego nad dziesiętnym w konstrukcji maszyn matematycznych oraz pętlą programową – **stałą się pierwszą w dziejach programistką**;



Maszyna analityczna splata algebraiczne wzory tak, jak maszyna Jacquarda tka kwiaty i liście.

- 1850 – George Boole opracował zasady algebry Boole'a;

Trochę historii …

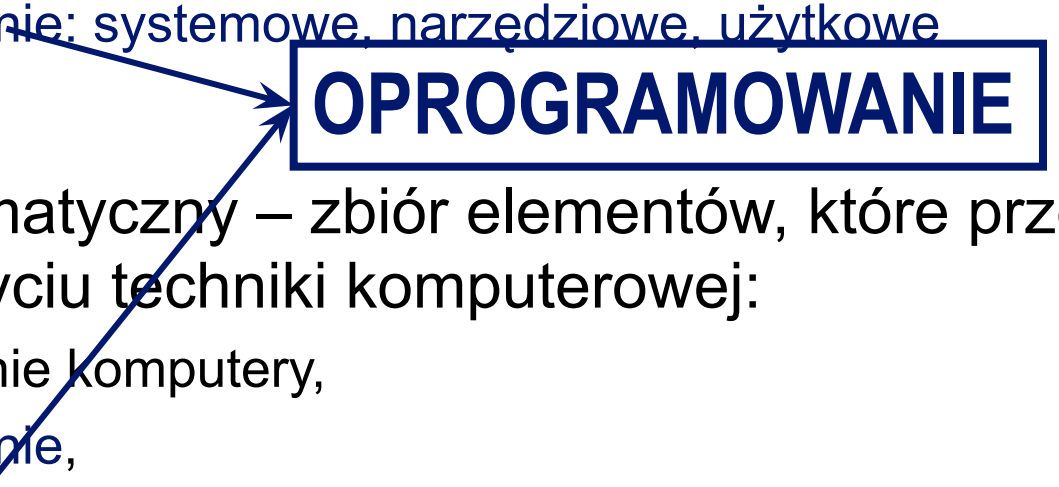
- 1946 – powstał ENIAC (Electronic Numerical Integrator and Computer), skonstruowany przez Johna W. Mauchly'ego i J. Presper Eckerta z amerykańskiego Ballistic Research Laboratory;
- **1967 – w Norweskim Centrum Obliczeniowym w Oslo powstał język Simula, uważany za przodka obiektowości;**
- 1972 - w Bell Laboratories opracowano język C;
- **1985 - Microsoft wypuścił na rynek Windows 1.0.**
- 1991 - Linus Torvalds z Unwersytetu Helsińskiego opracował odchudzoną wersję Unixa – Linux;
- 1996 - po wielkiej kampanii reklamowej Microsoft zaprezentował Windows 95;
- 1996 - … Era sieci globalnych, urządzeń programowalnych, komputerów przenośnych;

Trochę historii …

Komputer przenośny

Programowanie … oprogramowanie

- System komputerowy – układ współdziałania dwóch składowych: sprzętu komputerowego oraz oprogramowania o strukturze:
  - sprzęt,
  - oprogramowanie: systemowe, narzędziowe, użytkowe
  - użytkownicy;
- System informatyczny – zbiór elementów, które przetwarzają dane przy użyciu techniki komputerowej:
  - sprzęt – głównie komputery,
  - oprogramowanie,
  - zasoby osobowe, elementy organizacyjne (procedury organizacyjne, instrukcje robocze), elementy informacyjne (dane);



Programowanie … oprogramowanie

- Oprogramowanie – to programy komputerowe, ich dokumentacja, dane, pliki konfiguracyjne i pomocnicze …;
- **Program komputerowy** – ciąg instrukcji dla procesora prowadzący do realizacji założonego zadania, utworzony w **języku programowania** w procesie tworzenia programu (czyli w **programowaniu)** przez **programistę**;
- Gospodarki wszystkich rozwiniętych krajów zależą od oprogramowania … a jednocześnie…

  wytwarzanie oprogramowania jest poważną gałęzią gospodarki narodowej każdego rozwiniętego kraju;
- Czego wymagamy i wymaga się od nas?

  dobrego oprogramowania

Wymagania dla dobrego oprogramowania

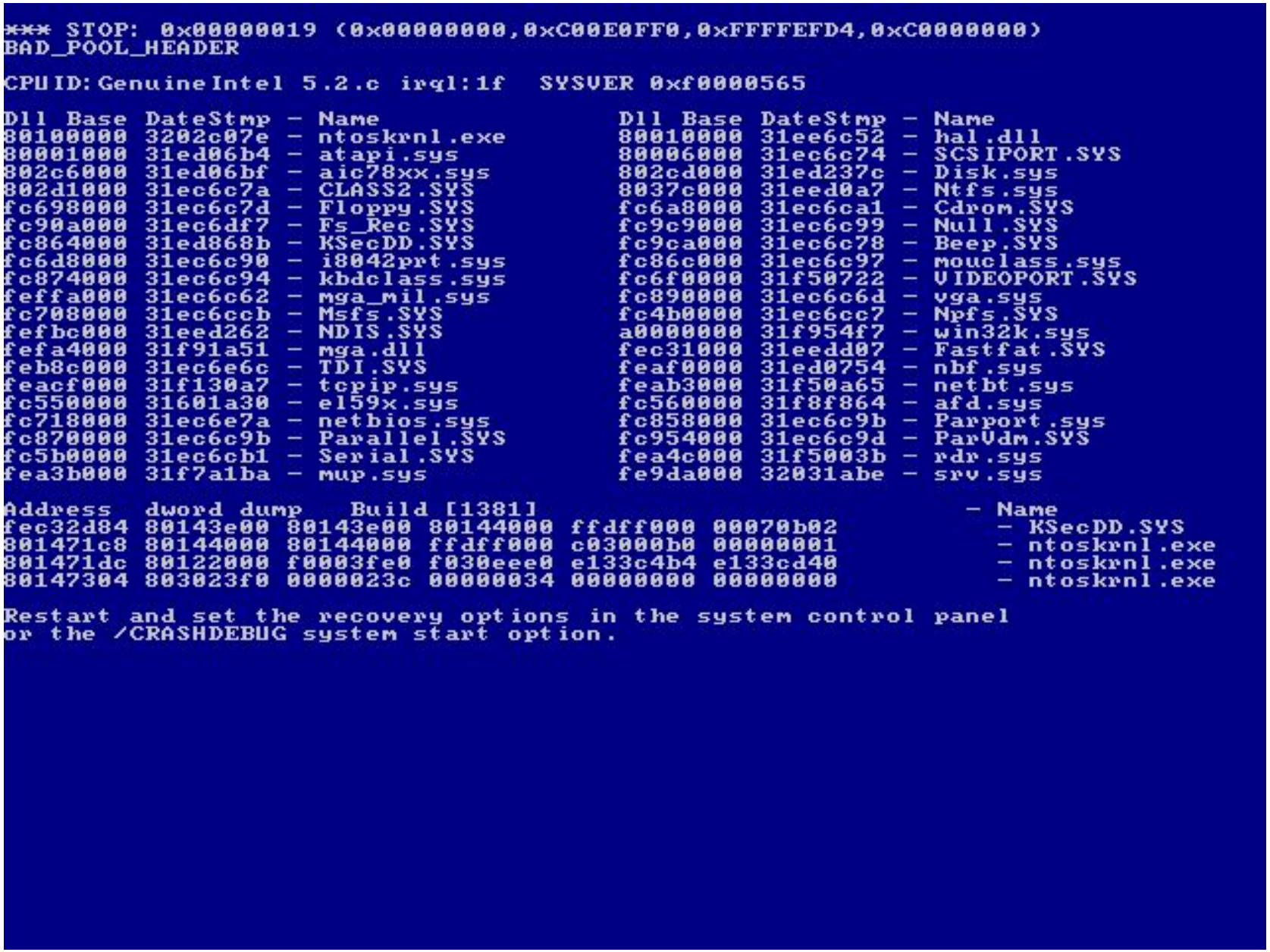
- Dobre oprogramowanie powinno zapewniać:
  - użyteczność - dostępność oczekiwanych usług,
  - niezawodność,
  - efektywność,
  - bezpieczeństwo zasobów (w tym wyników pracy),
  - ochrona (w tym przed zewnętrznymi intruzami),
  - ergonomia,
  - wielokrotne wykorzystanie,
  - przenośność,
  - podatność na pielęgnowalnie;
  - efektywność kosztowa - opłacalność;

Programowanie … oprogramowanie

- Język programowania powinien:
  - wspomagać wierne odwzorowanie rzeczywistości,
  - wymuszać i wspierać logiczną organizację programu,
  - tworzyć kod przenośny, czytelny i zrozumiały,
  - utrudniać popełnianie błędów algorytmicznych,
  - samodokumentować program;

- A jaką mamy rzeczywistość?

Wymagania dla dobrego oprogramowania

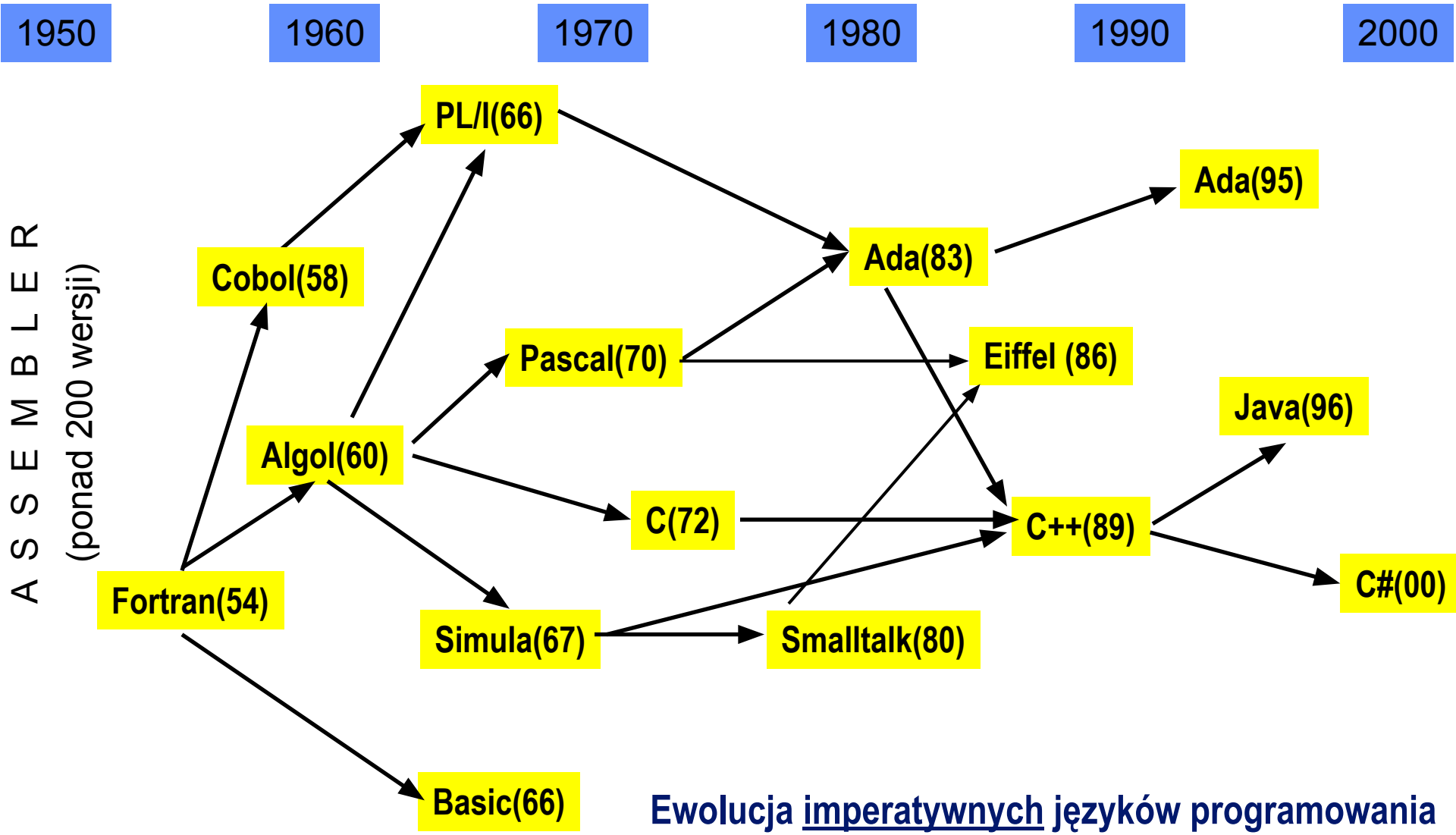


Wymagania dla dobrego oprogramowania

Ewolucja technik programowania

- Obecnie na świecie jest kilka tysięcy języków programowania;
- Już w 1995 roku na *comp.lang.misc* zanotowano ponad 2300 odwołań do różnych języków;
- Klasyfikacja języków programowania:
  - imperatywne (imperative),
  - funkcjonalne, proceduralne (functional),
  - logiki (logic),
  - obiektowe, obiektowo zorientowane (object-oriented);

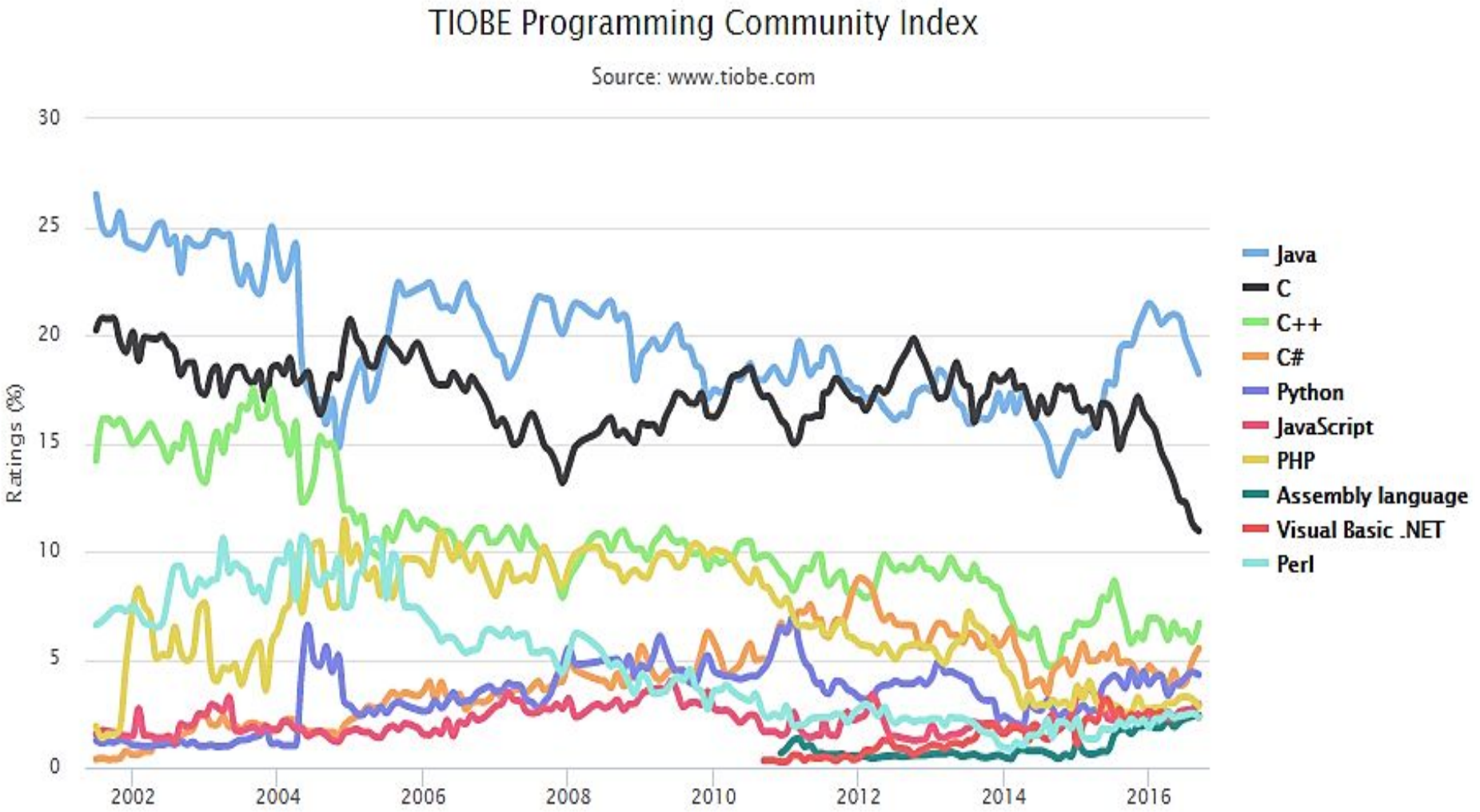
Ewolucja technik programowania



Ewolucja imperatywnych języków programowania

Ewolucja technik programowania

http://www.tiobe.com



Podsumowanie

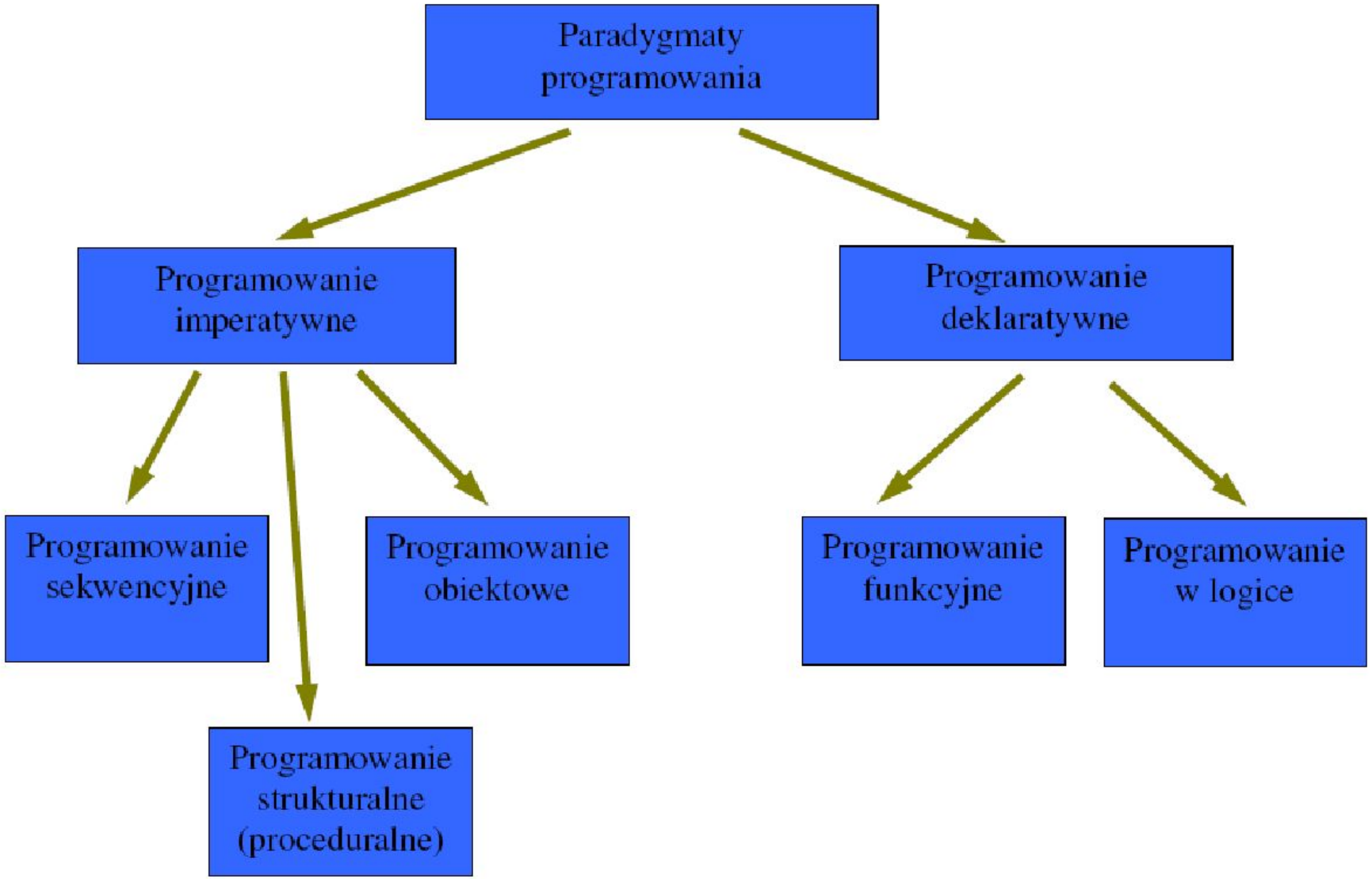
http://www.tiobe.com

Programming Language	2016	2011	2006	2001	1996	1991	1986
Java	1	1	1	3	14	-	-
C	2	2	2	1	1	1	1
C++	3	3	3	2	2	2	5
C#	4	5	6	11	-	-	-
Python	5	6	7	24	23	-	-
PHP	6	4	4	8	-	-	-
JavaScript	7	9	8	7	19	-	-
Visual Basic .NET	8	30	-	-	-	-	-
Perl	9	8	5	4	3	-	-
Ruby	10	10	18	32	-	-	-
Lisp	27	12	12	15	7	5	3
Ada	28	16	15	16	6	3	2

Ewolucja technik programowania

- Paradygmat programowania (ang. programming paradigm) – zaakceptowany powszechnie wzorzec programowania definiujący sposób patrzenia programisty na przepływ sterowania i wykonywanie programu komputerowego;
- Różne języki programowania mogą wspierać różne paradygmaty programowania – najczęściej dla języka istnieje jeden dominujący paradygmat, choć np. C++ posiada elementy programowania proceduralnego, obiektowego oraz uogólnionego (generycznego);
- Powszechnie uznane paradygmaty programowania:
  - **programowanie imperatywne**
  - **programowanie strukturalne**
  - programowanie proceduralne
  - programowanie funkcyjne
  - **programowanie obiektowe**
  - programowanie uogólnione (generyczne)
  - programowanie sterowane zdarzeniami
  - programowanie logiczne (np. Prolog)
  - programowanie aspektowe (np. AspectJ)
  - programowanie deklaratywne
  - programowanie agentowe
  - programowanie modularne

Ewolucja technik programowania



Ewolucja technik programowania

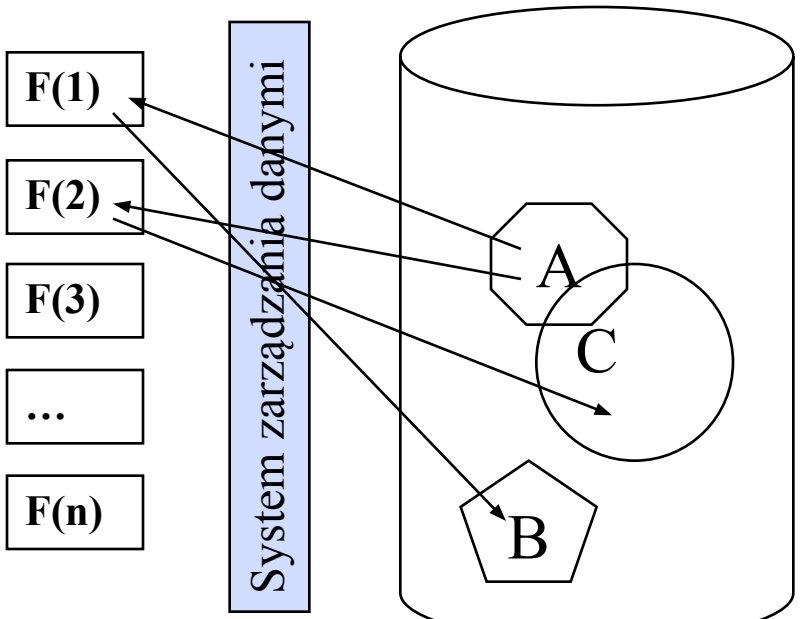
- Programowanie imperatywne – proces wykonywania programu jest sekwencją instrukcji zmieniających stan programu;
  - Programy imperatywne składają się z ciągu komend (żądania czynności) do wykonania przez komputer;
  - Przykłady języków programowania FORTRAN, ALGOL, Pascal, C i Ada;
- Programowanie strukturalne – opiera się na podziale kodu źródłowego programu na procedury i hierarchicznie ułożone bloki z wykorzystaniem struktur kontrolnych w postaci instrukcji: sekwencji, wyboru i pętli;
- Programowanie obiektowe – programy definiuje się za pomocą obiektów – elementów łączących stan (czyli dane) i zachowanie (czyli procedury, metody) – komunikujących się ze sobą w celu wykonywania zadań;
- Programowanie logiczne – odmiana programowania deklaratywnego, w której program to zestaw zależności, a obliczenia są dowodem pewnego twierdzenia w oparciu o te zależności;

Ewolucja technik programowania

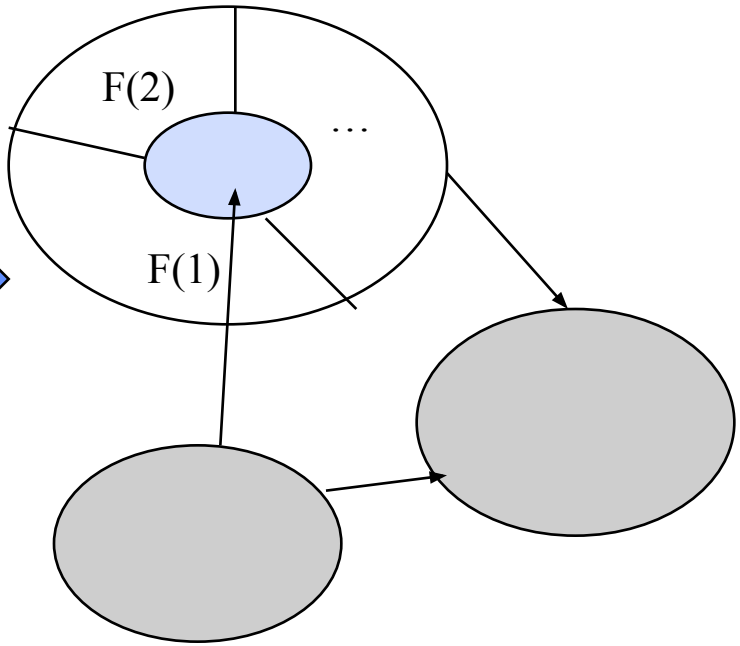
- programowanie proceduralne:
  - program = seria procedur, działających na danych;
  - dane całkowicie odseparowane od procedur;
- programowanie strukturalne:
  - rozszerzenie programowania proceduralnego;
  - główna idea: „dziel i rządź” – od ogółu do szczegółu;
- programowanie obiektowe:
  - główne zadanie to modelowanie „obiektów” a nie „danych”;
  - łączy w logiczną całość dane oraz manipulujące nimi funkcje;
  - wspiera konstruowanie systemów od szczegółu do ogółu;

Ewolucja technik programowania

- Od programowania strukturalnego do obiektowego…



Architektura systemu komputerowego Von Neumanna



Architektura systemu obiektowego

Ewolucja technik programowania

- Programowanie obiektowe:
  - główne zadanie to modelowanie „obiektów” (tzn. rzeczy, zjawisk), a nie „danych.”;
  - modelowanymi obiektami mogą być zarówno elementy programowe (np. przyciski, pola list), jak i obiekty świata rzeczywistego, np. samoloty, organizmy, procesy;
  - łączy w logiczną całość dane oraz manipulujące nimi funkcje;
  - wspiera konstruowanie systemów od szczegółu do ogółu – zysk:
    - umożliwia ponowne wykorzystanie komponentów;
    - ułatwia modyfikowanie oprogramowania;

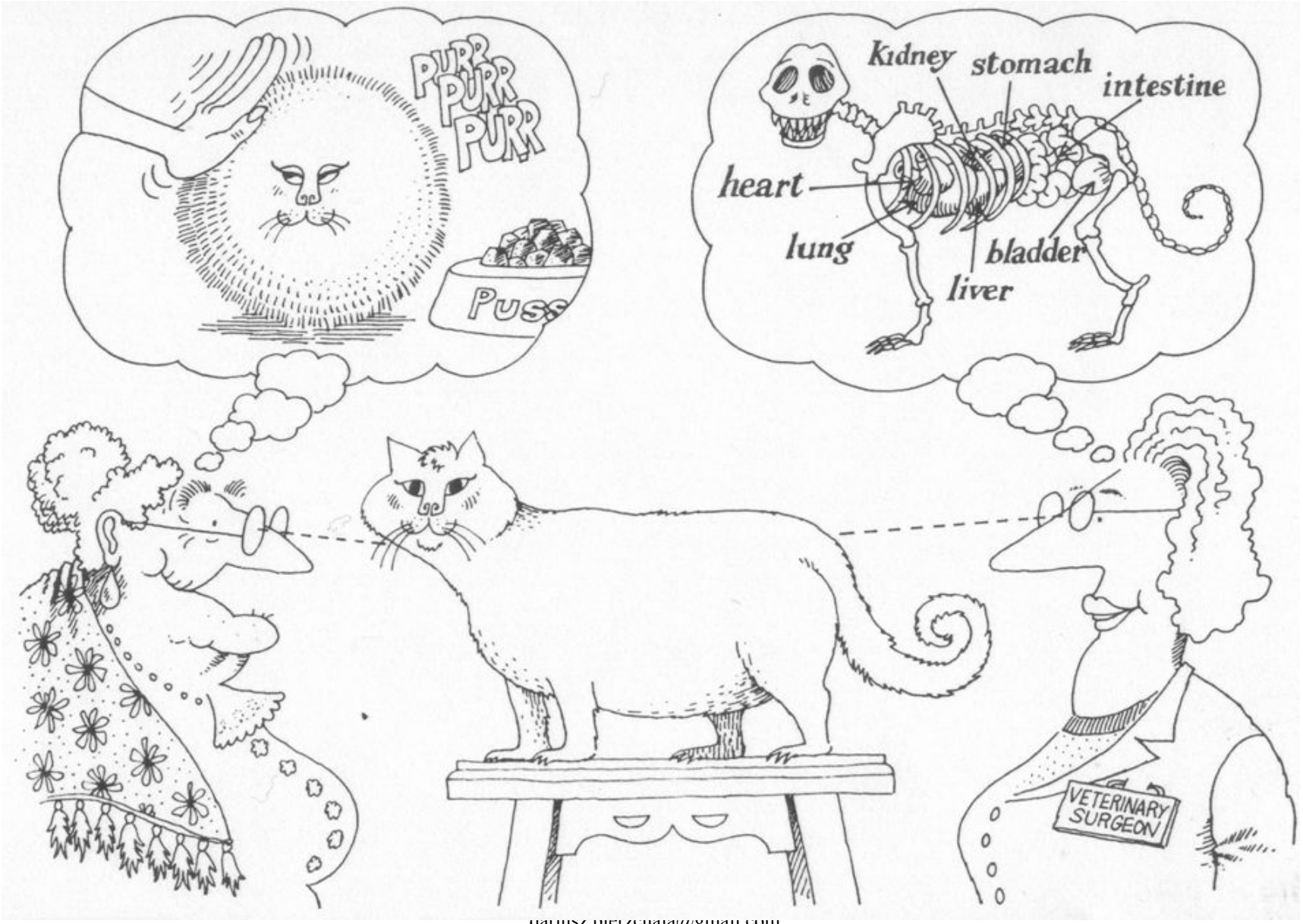
Koncepcja obiektowości

- obiektowość - cecha naturalnego postrzegania świata - analiza otoczenia poprzez relacje między obserwatorem a otaczającymi obiektami;
- świat jest złożony - składa się z wielu funkcjonujących obiektów, pozostających w pewnych relacjach względem siebie;
- obiektami są ludzie, państwa, domy, samochody, ale także płace, zadania, decyzje…;
- obiektowość jest podstawą obiektowej analizy, projektowania i programowania systemów;

Koncepcja obiektowości

- Paradygmat obiektowy (*podstawowy styl, techniki oraz wspomagające je konstrukcje językowe*) :
  - abstrakcja
  - hermetyzacja (kapsułkowanie)
  - dziedziczenie
  - tożsamość instancji klas (obiektów)
  - polimorfizm
  - komunikaty
  - klasy generyczne

Koncepcja obiektowości



Koncepcja obiektowości

- Obiekt:
  - podstawowa jednostka konstrukcyjna;
  - konkretny lub abstrakcyjny byt (wyróżnialny w modelowanej rzeczywistości) posiadający nazwę, jednoznaczną identyfikację, określone granice, atrybuty i inne własności;
  - posiada następujące rodzaje właściwości i odpowiedzialności:
    - Atrybuty – reprezentują stan obiektu i związki z innymi obiektami, np. kolor, rozmiar, przynależność…
    - Procedury (usługi, metody) – operacje, które obiekt może wykonywać, np. przemieszczanie, całkowanie, wyznaczanie stanu konta…
    - Zasady – niezmiennicze reguły określające widzialność obiektu i sposób powiązania z innymi obiektami.
  - *abstrakcyjny typ danych, korzystający z dostępnych w języku programowania typów danych wraz z operacjami, które mogą być wykonywane na tych typach;*

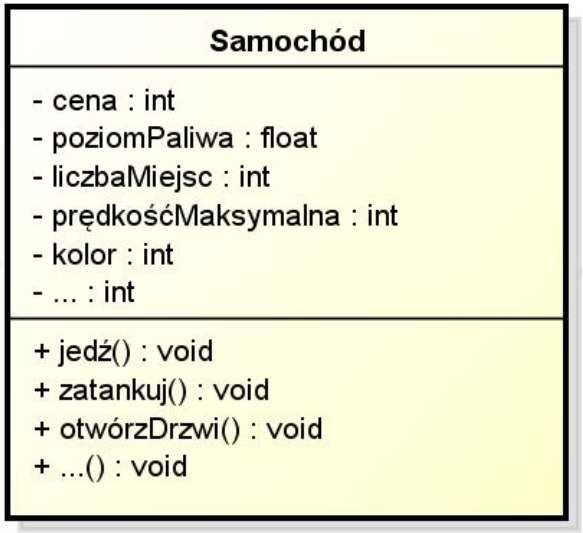
Koncepcja obiektowości

- Klasa:
  - zbiór obiektów, mających wspólne atrybuty i metody;
  - wzorzec dla konkretnych egzemplarzy klasy – obiektów;
  - instensja typu obiektowego - definicja pojęcia, pewna koncepcja, idea stosująca się do określonej grupy obiektów, np. środek umożliwiający transport ludzi i rzeczy;
  - ekstensja typu obiektowego - zbiór konkretnych typów (klas, pojęć), np. pojazdy lądowe, statki powietrzne i wodne;
  - *wyrażenie językowe specyfikujące budowę obiektów, dozwolone operacje na obiektach, ograniczenia dostępu, wyjątki, itd.*

Koncepcja obiektowości

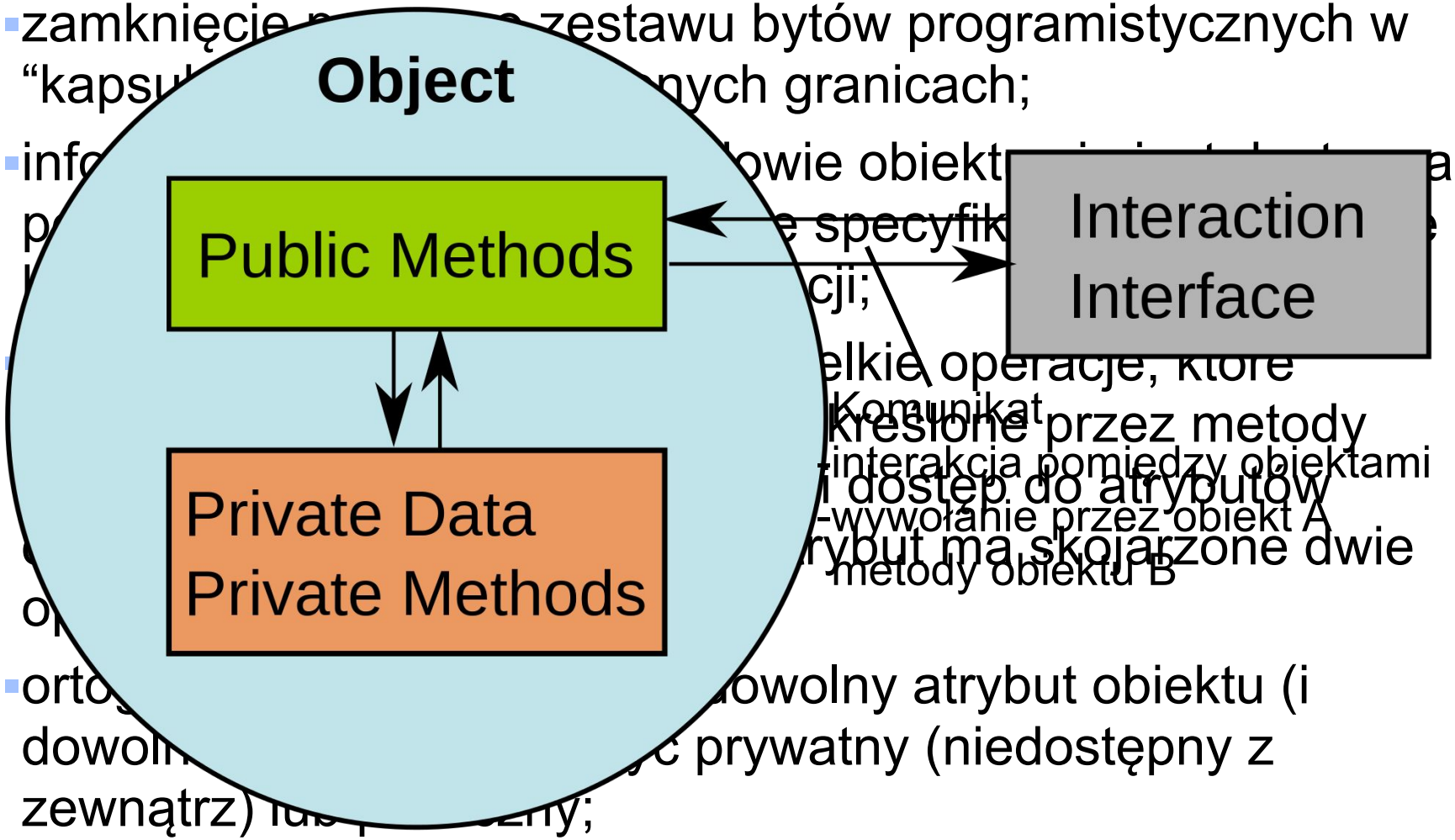
Klasy i ich instancje

Jedna klasa może mieć wiele instancji, które różnić się mogą wartościami atrybutów. Każde wystąpienie klasy nazywane jest Instancją Klasy lub Obiektem.



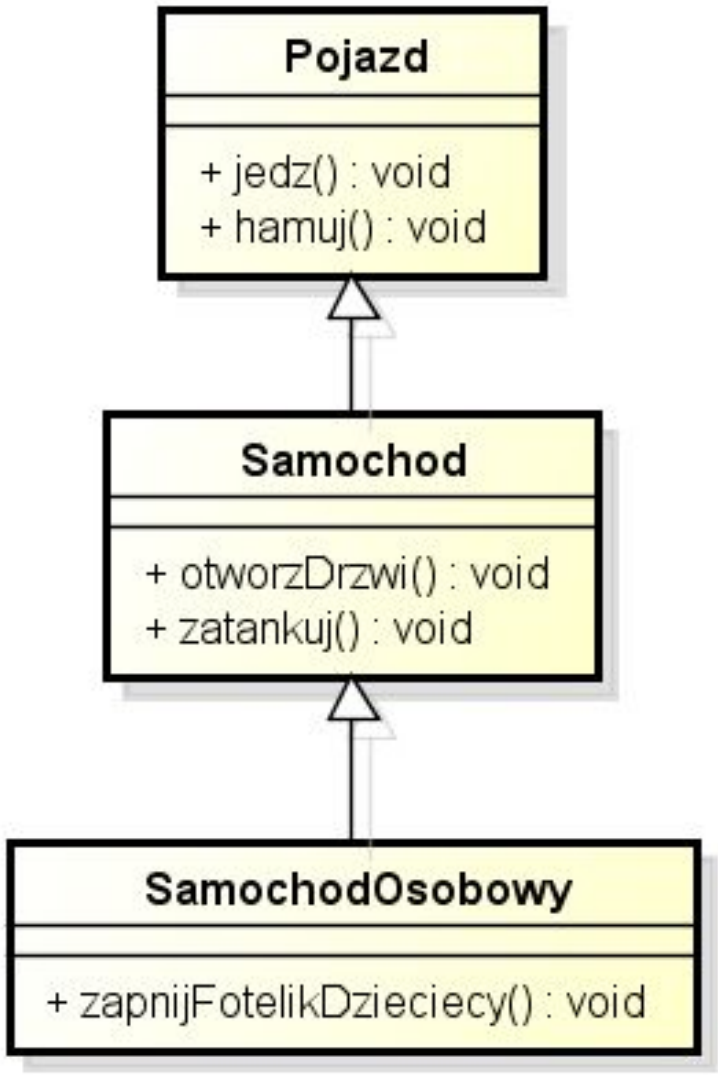
Koncepcja obiektowości

- Hermetyzacja (kapsułkowanie, enkapsulacja)
  - zamknięcie zestawu bytów programistycznych w “kapsu nych granicach;
  - info owie obiekt a p specyfik e l cji;
  - elkie operacje, które kreślone przez metody i dostęp do atrybutów rybut ma skojarzone dwie o
  - orto owolny atrybut obiektu (i dowoln c prywatny (niedostępny z zewnątrz) lub



Koncepcja obiektowości

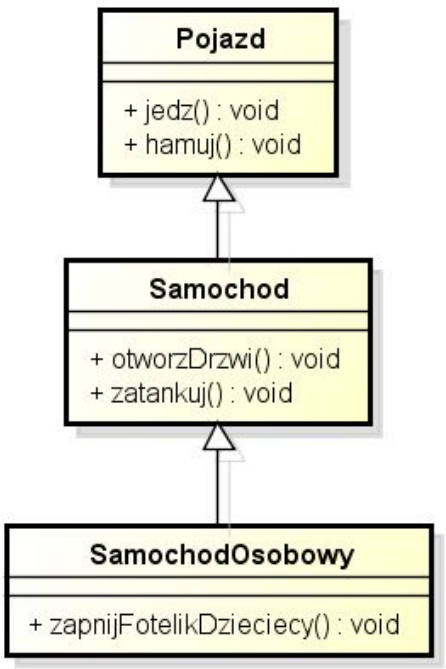
- Dziedziczenie
  - związek pomiędzy klasami
    przekazywanie cech (defini
    nadklasy do jej podklas;
  - np. obiekt klasy Samochód
    (definicje atrybutów, metod
  - dziedziczenie jest podstaw
    sprzyjającym ponownemu
  - formy dziedziczenia:
    - statyczne
    - dynamiczne,
    - jednostkowe,
    - wielokrotne;



Koncepcja obiektowości

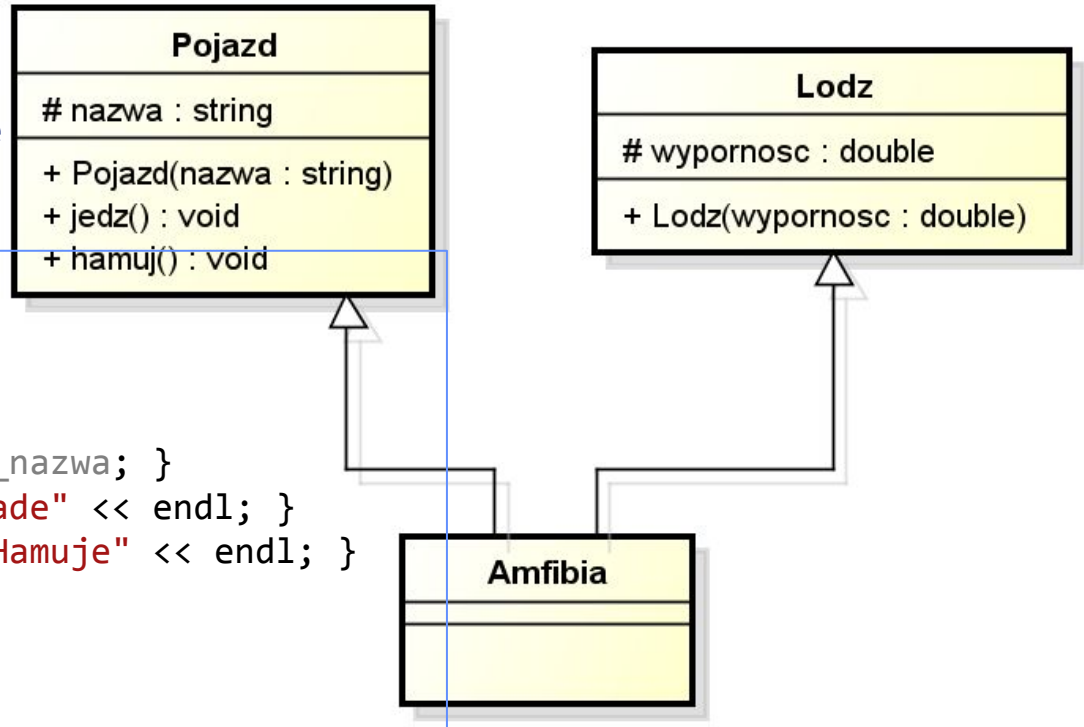
```
class Pojazd {
public:
    virtual void jedz() { cout << "Jade" << endl; }
    virtual void hamuj() { cout << "Hamuje" << endl; }
};

class Samochod : public Pojazd {
public:
    virtual void otworzDrzwi() { cout << "Otwieram
drzwi" << endl; }
    virtual void zatankuj() { cout << "Tankuje" <<
endl; }
};

class SamochodOsobowy : public Samochod {
public:
    virtual void zapnijFotelikDzieciecy() { cout <<
"Zapinam fotelik" << endl; }
};
```

```
int main() {

  SamochodOsobowy samochodOsobowy;
  Samochod& samochod = samochodOsobowy;
  Pojazd& pojazd = samochodOsobowy;

  samochodOsobowy.hamuj();
  samochod.hamuj();
  pojazd.hamuj();

  return 0;
}
```



Koncepcja obiektowości

Dziedziczenie wielobazowe

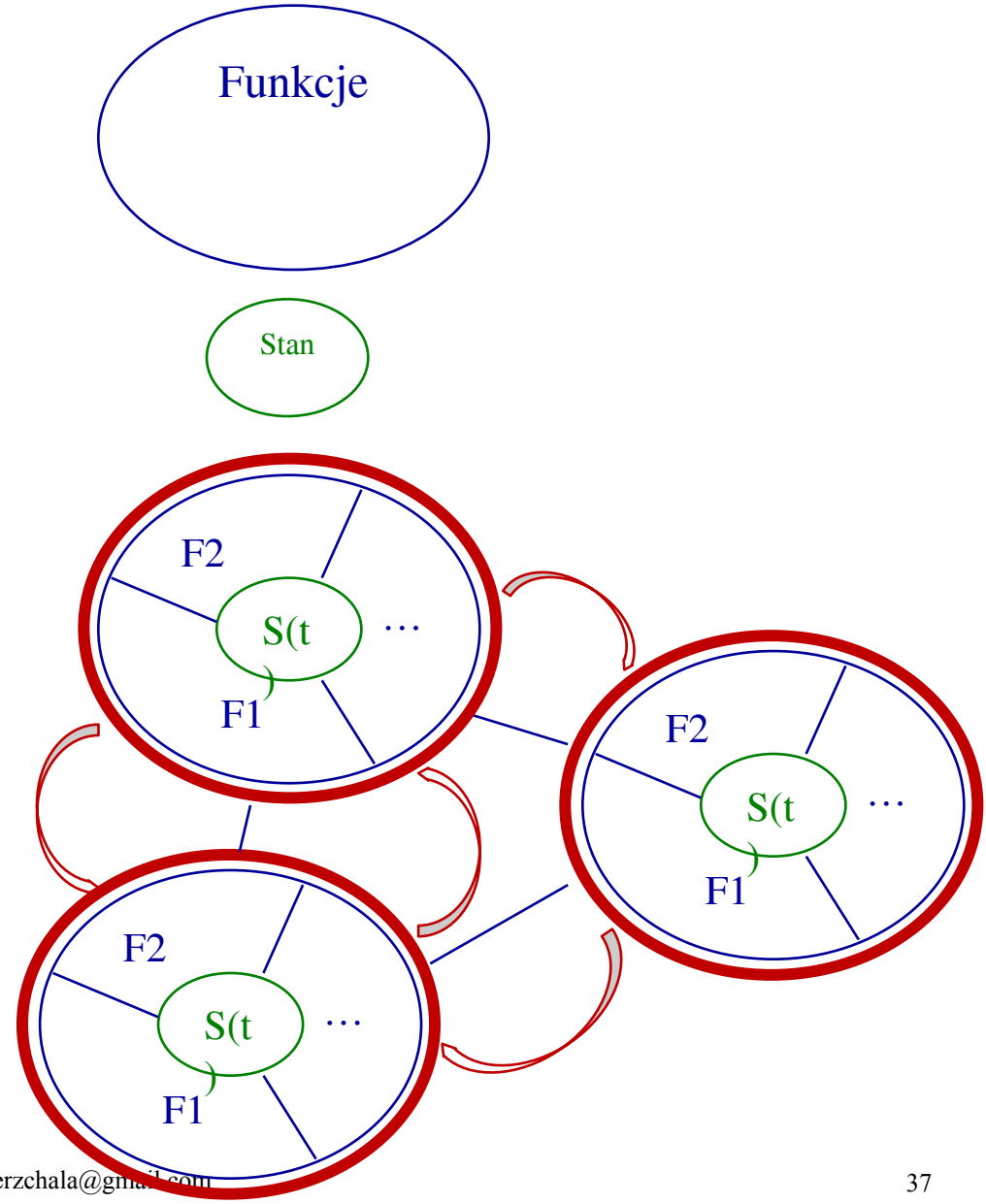



```
class Pojazd {
protected:
    string nazwa;
public:
    Pojazd(string _nazwa) { nazwa = _nazwa; }
    virtual void jedz() { cout << "Jade" << endl; }
    virtual void hamuj() { cout << "Hamuje" << endl; }
};

class Lodz {
protected:
    double wypornosc;
public:
    Lodz(double _wypornosc){ wypornosc = _wypornosc; }
};

class Amfibia : public Pojazd, public Lodz {
    ...
};
```

Ewolucja technik programowania

- Funkcje programu
  - *Np. obliczanie wartości X*
- Stany programu (dane)
  - *Np. X = [x_1, x_2, …, x_N]*
- Klasy i obiekty
  - relacje
- Agenty
  - autonomia
  - komunikacja
  - percepcja

agent, agenty – forma bezosobowa



Ewolucja technik programowania

http://www.tiobe.com

Category	Ratings Feb 2012	Delta Feb 2011
Object-Oriented Languages	57.6%	+0.4%
Procedural Languages	36.3%	-1.0%
Functional Languages	4.2%	-0.1%
Logical Languages	1.9%	+0.7%

Category	Ratings February 2009	Delta February 2008
Object-Oriented Languages	55.9%	+1.3%
Procedural Languages	39.9%	-2.7%
Functional Languages	3.0%	+1.2%
Logical Languages	1.2%	+0.2%

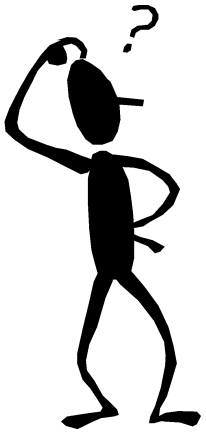
Ewolucja technik programowania

- Zestawienie cech wybranych języków programowania

Language	Imperative	Object-oriented	Functional	Procedural	Generic	Reflective	Event-driven	Other paradigm(s)
Ada	Yes	Yes		Yes	Yes			concurrent, distributed
C	Yes			Yes				
C++	Yes	Yes	Yes	Yes	Yes			
C#	Yes	Yes	Yes	Yes	Yes	Yes	Yes	structured, concurrent
Java	Yes	Yes	Yes	Yes	Yes	Yes	Yes	concurrent
Perl	Yes	Yes	Yes	Yes	Yes	Yes		
PHP	Yes	Yes	Yes	Yes		Yes		
Python	Yes	Yes	Yes	Yes		Yes		aspect-oriented

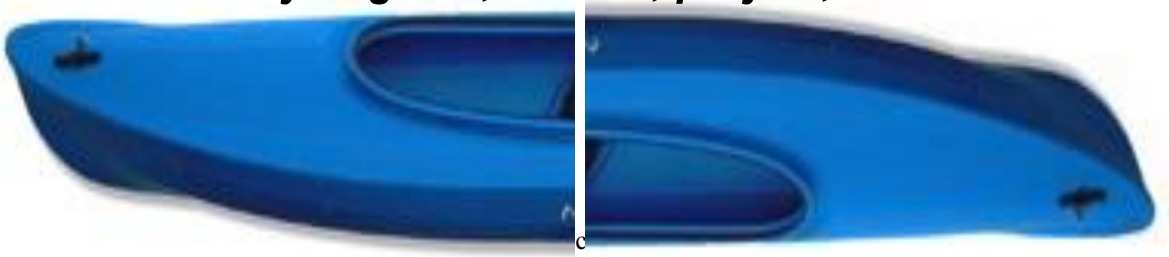
Ewolucja technik programowania

- Pamiętać jednak należy, że:
  - Programowanie nie zaczyna się i nie kończy na przekładaniu pomysłów z głowy na polecenia języka programowania – po drodze są różne etapy/fazy/dyscypliny/zadania;
  - Podobnie jak budowa domu – potrzebny są:
    - pomysł, prototyp/projekt, murarz, aranżator wnętrz, wyposażenie, użytkownik, sponsor;
  - Odpowiada to w programowaniu pojęciom:
    - koncepcja, makieta/projekt, programista, grafik GUI, dane, użytkownik i sponsor.
  - Kodowanie (programowanie) jest tylko jedną z faz/etapów;



Programowanie…

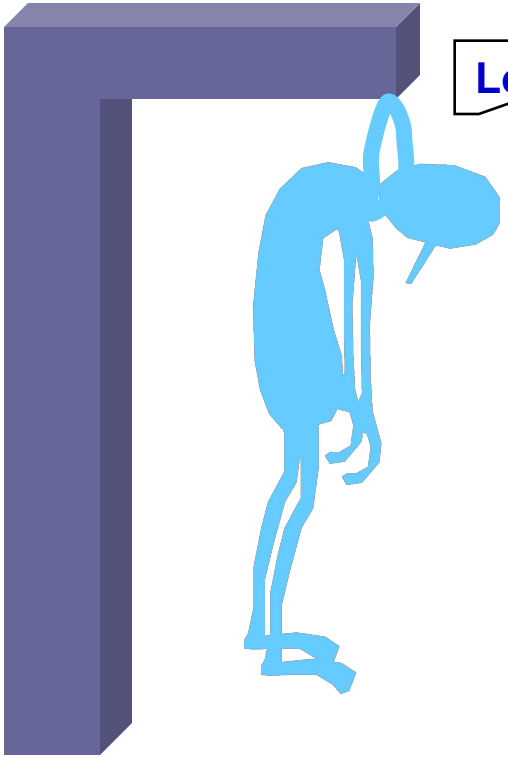
…ale także wymagania, analiza, projekt, testowanie



Geneza inżynierii oprogramowania

Kryzys oprogramowania

- Od początku lat 60-tych trwa walka z syndromem LOOP;



***L**ate* (późno)

***O**ver budget* (przekroczony budżet)

***O**vertime* (nadgodziny)

***P**oor quality* (kiepska jakość)

- 1968, 1969 - konferencje sponsorowane przez NATO w Garmisch i Rzymie;

Kryzys oprogramowania

Lata	Oprogramowanie	Kryzys	Stan ‘Inż. Opr.’
1950 –1960	Tworzone dla siebie	Błędy własne	Brak
1960 –1970	Duże systemy	Początek kryzysu, kosztowne błędy	Powstaje,
1970 –1990	Powszechnie dostępne komputery, duże i masowo dystrybuowane oprogr.	Błędy powszechnie występujące	Rozkwit inżynierii
1990 – …	Uzależnienie gospodarki i wielu dziedzin życia od komputerów	Kryzys trwa w dalszym ciągu	Rozwój trwa – nowe teorie i praktyki

Kryzys oprogramowania

- Walka z kryzysem oprogramowania:
  - Usuń przyczyny -> wyeliminujesz zauważone symptomy;
  - Stosowanie technik i narzędzi ułatwiających pracę nad złożonymi systemami;
  - Korzystanie z metod wspomagających analizę nieznanych problemów oraz ułatwiających wykorzystanie wcześniejszych doświadczeń;
  - Usystematyzowanie procesu wytwarzania oprogramowania, tak, aby ułatwić jego planowanie i monitorowanie;
  - Wytworzenie wśród producentów i nabywców przekonania, że budowa dużego systemu wysokiej jakości jest zadaniem wymagającym profesjonalnego podejścia;

Co to jest inżynieria oprogramowania?

- Jest to dziedzina inżynierii, która obejmuje wszystkie aspekty tworzenia oprogramowania od fazy początkowej do jego pielęgnacji;
- Inżynieria oprogramowania jest wiedzą empiryczną, syntezą doświadczenia wielu ośrodków zajmujących się budową oprogramowania;
- Informatyka obejmuje teorie i podstawowe zasady działania komputerów a inżynieria oprogramowania obejmuje praktyczne problemy konstruowania oprogramowania;
- Zajmuje się efektywnymi teoriami, metodami i narzędziami związanymi z procesem wytwarzania oprogramowania;
- Zastosowanie systematycznego, zdyscyplinowanego, ilościowego podejścia do wykonywania, wykorzystywania i konserwowania oprogramowania [IEEE 93];

Co to jest inżynieria oprogramowania?

- Metodyki:
  - Strategia postępowania oparta na doświadczeniach i heurystykach oraz formalnych elementach;
  - Zbiór reguł, zasad, metod, technik i narzędzi wykorzystywany do realizacji funkcji planowania, zarządzania, projektowania i realizacji przedsięwzięć informatycznych;
- Metody:
  - uporządkowane procedury budowy oprogramowania, które poprzez zdefiniowane techniki umożliwią posługiwanie się narzędziami w celu poznania rzeczywistości oraz modelowania jej zgodnie z przyjętym celem działania;

Co to jest inżynieria oprogramowania?

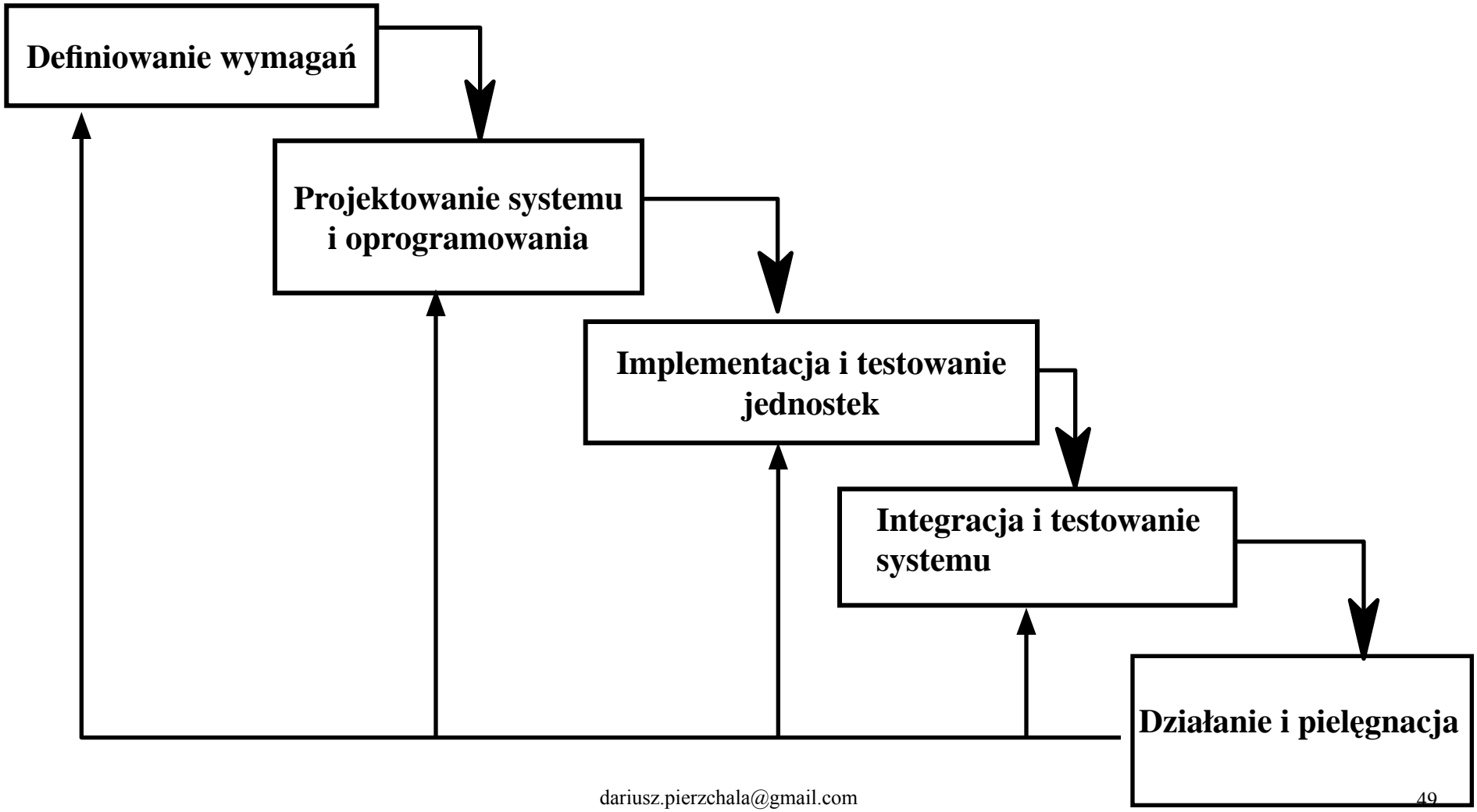
- Techniki:
  - szczegółowo określone sposoby (z wykazem czynności) posługiwania się środkami, w tym narzędziami, z danej metody dla osiągnięcia założonego celu;
- Narzędzia CASE:
  - przeznaczone do wspomagania rutynowych czynności, takich jak praca nad diagramami projektowymi, sprawdzanie poprawności diagramów oraz śledzenie wykonanych testów;
  - Upper-CASE (dla wszystkich etapów);
  - Lower-CASE (wspomaganie implementacji);
  - Integrated-CASE (wszystkie fazy);

Proces tworzenia oprogramowania

- Jest to zbiór czynności i związanych z nimi wyników, które prowadzą do powstania produktu programowego;
- Zasadnicze czynności wspólne dla wszystkich procesów:
  - ***Specyfikowanie oprogramowania***
    - Funkcjonalność oprogramowania i ograniczenia jego działania muszą być zdefiniowane;
  - ***Projektowanie i implementowanie oprogramowania***
    - Oprogramowanie, które spełnia specyfikację, musi być stworzone;
  - ***Zatwierdzenie oprogramowania***
    - Wytworzone oprogramowanie musi spełniać oczekiwania klienta;
  - ***Ewolucja oprogramowania***
    - Oprogramowanie musi ewoluować, aby spełniać zmieniające się potrzeby użytkowników;

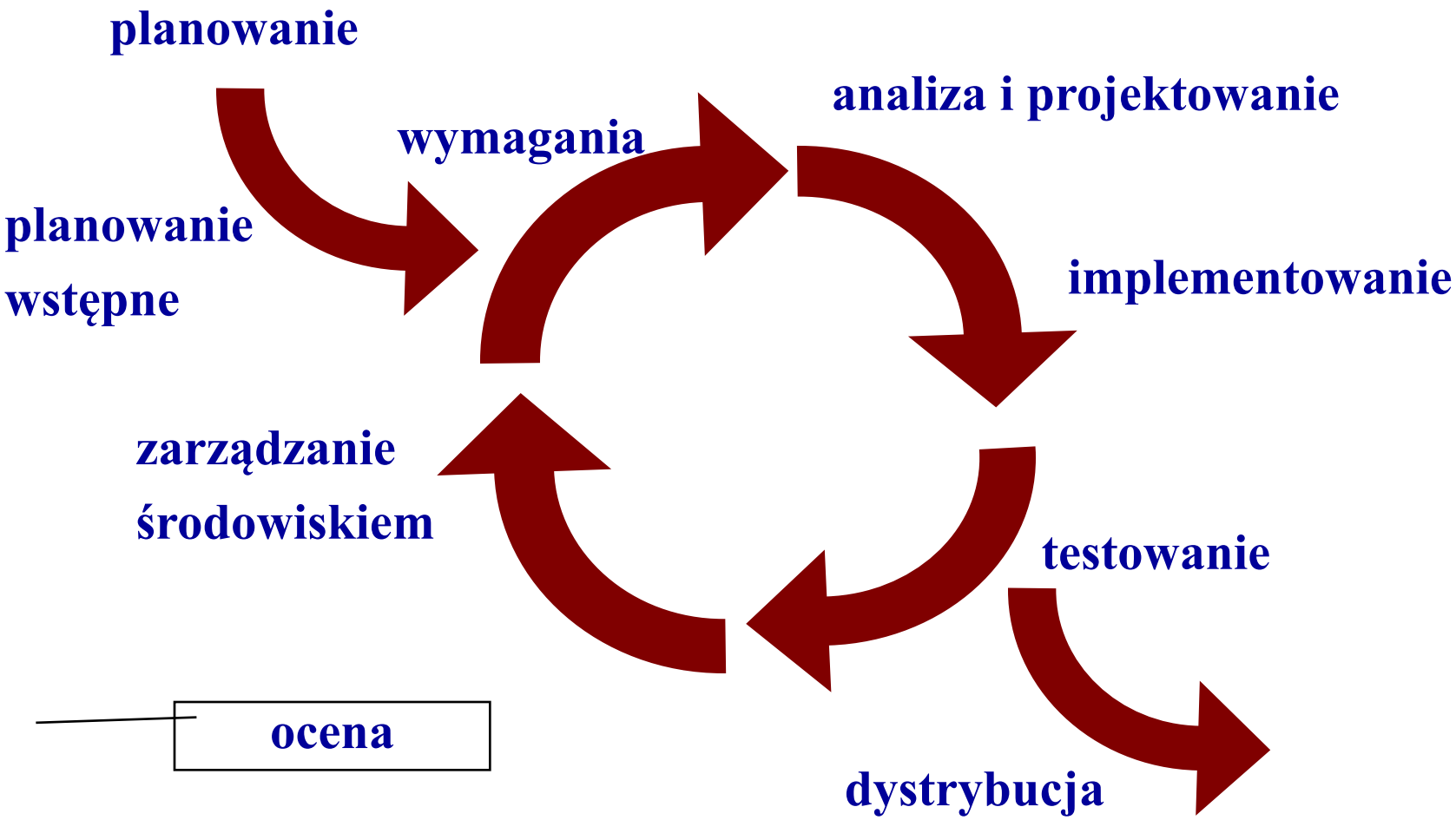
Modele procesu tworzenia oprogramowania

- Model kaskadowy (wodospadowy) - *waterfall model*



Modele procesu tworzenia oprogramowania

- Tworzenie iteracyjne



Modele procesu tworzenia oprogramowania

- Ewolucja inżynierii oprogramowania - podsumowanie:

Języki:	Assembly -> Fortran/COBOL -> Simula -> C++ -> Java / C#
Platformy:	prosty HW -> BIOS -> OS -> Middleware -> Domain-specific
Modele (cykle):	Waterfall -> Spiral -> Iterative -> Agile
Architektura:	Procedural -> Object Oriented -> Service Oriented
Narzędzia:	Early tools -> CLE -> IDE -> XDE -> CDE
Org. pracy:	Individual -> Workgroup -> Organization

Proces tworzenia oprogramowania

- Zbiór czynności i związanych z nimi wyników, które prowadzą do powstania produktu programowego;
- Zasadnicze czynności wspólne dla wszystkich procesów:
  - ***Specyfikowanie oprogramowania***
    - Funkcjonalność oprogramowania i ograniczenia jego działania muszą być zdefiniowane;
  - ***Projektowanie i implementowanie oprogramowania***
    - Oprogramowanie, które spełnia specyfikację, musi być stworzone;
  - ***Zatwierdzenie oprogramowania***
    - Wytworzone oprogramowanie musi spełniać oczekiwania klienta;
  - ***Ewolucja oprogramowania***
    - Oprogramowanie musi ewoluować, aby spełniać zmieniające się potrzeby użytkowników;

Czynności procesu tworzenia oprogramowania

- Projektowanie i implementowanie wymagań:
  - Metody projektowania:
    - Stosowanie metod strukturalnych i obiektowych, które są zbiorami notacji i porad dla projektantów oprogramowania;
    - Ich użycie polega głównie na opracowaniu graficznych modeli systemu oraz opisów tekstowych wg ustalonych szablonów;
    - Metody strukturalne obejmują np.:
      - modele przepływu danych,
      - modele encja-związek;
    - **Niezwykle popularne są metody obiektowe, w tym np.:**
      - **modele klas, dynamiki,**
      - **modele architektury;**
  - Niestety wciąż jeszcze projektowanie jest działaniem *ad hoc*, gdzie wymagania zapisuje się w języku naturalnym;

Czynności procesu tworzenia oprogramowania

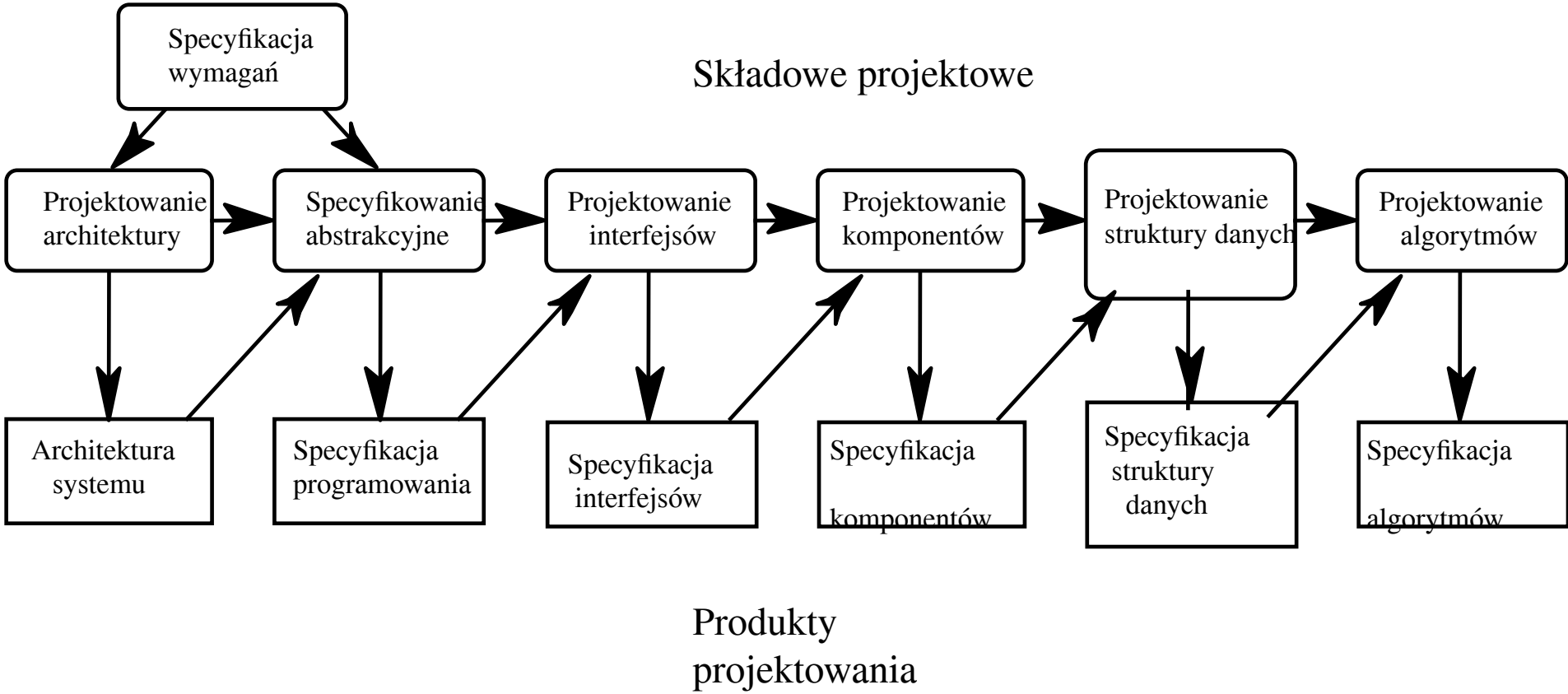
- Projektowanie i implementowanie oprogramowania:
  - Celem fazy określania wymagań jest udzielenie odpowiedzi na pytanie: co i przy jakich ograniczeniach system ma robić?
  - Faza analizy:
    - Jest łącznikiem między specyfikacją wymagań a projektowaniem;
    - Ustalane są wszystkie uwarunkowania dziedzinowe, organizacyjne i inne, które mogą wpłynąć na decyzje projektowe i na realizację wymagań;
    - Celem fazy analizy jest udzielenie odpowiedzi na pytanie: Jak system ma działać?
    - wynikiem jest logiczny model systemu, opisujący sposób realizacji przez system postawionych wymagań, lecz abstrahujących od szczegółów implementacyjnych;

Czynności procesu tworzenia oprogramowania

- Projektowanie i implementowanie oprogramowania:
  - Faza projektowania oprogramowania:
    - opis struktury oprogramowania, które ma być zaimplementowane, opis danych, które są częścią systemu, opis interfejsów między komponentami systemu i użytych algorytmów;
    - Celem fazy projektowania jest udzielenie odpowiedzi na pytanie: Jak system ma być zaimplementowany?
    - faza projektowania może obejmować opracowanie wielu modeli systemu na różnych poziomach abstrakcji;
    - Wynikiem jest szczegółowy opis sposobu implementacji;
  - Faza implementowania w tworzeniu oprogramowania to proces przekształcania specyfikacji systemu w działający system;

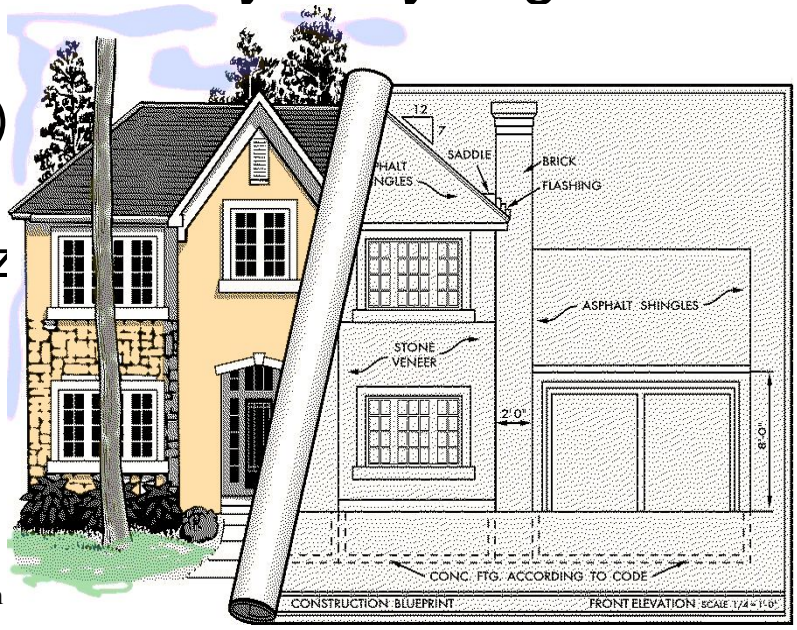
Czynności procesu tworzenia oprogramowania

- Projektowanie i implementowanie wymagań:
  - Charakterystyczne składowe analizy i projektowania:



Projektowanie architektury systemu

- System informatyczny jest złożoną konstrukcją, której stopień skomplikowania zależy od złożoności architektury;
- Wielkie systemy są zwykle podzielone na podsystemy, które oferują pewien zbiór powiązanych ze sobą interfejsów;
- Wymagane jest projektowanie architektoniczne, którego wynikiem jest opis architektury oprogramowania;
- Urzeczywistnienie pomysłów architektonicznych wymaga:
  - idei (pomysł, cel),
  - planów (architektura, zagospodarowanie)
  - wiedzy (metody, techniki),
  - zasobów (materiały, narzędzia, czas, ludz
  - nadzoru i pielęgnacji we wszystkich etapach życia (projekt, budowa, użytkowanie, wycofanie);



dariusz.pierzchala@gmail.com

Projektowanie architektury systemu

- Modele obiektowe:
  - Model obiektowy architektury systemu dzieli system na zbiór luźno uzależnionych od siebie obiektów, z dobrze zdefiniowanymi interfejsami.
  - Obiekty korzystają z usług oferowanych przez inne obiekty.
  - Podział obiektowy uwzględnia klasy obiektów, ich atrybuty i operacje.

Metodyki strukturalne a obiektowe

- Metodyki strukturalne:
  - metodyki strukturalne są dojrzałe, lecz mogą nie być adekwatne do współczesnych tendencji wytwarzania systemów informatycznych;
  - wadą metodyk strukturalnych są trudności w zintegrowaniu modeli;
- Metodyki obiektowe:
  - intuicyjne podejście do modelowania;
  - relatywnie młode i szybko zmieniające się;
  - wprowadzają narzuty implementacyjne;
  - dobre dla systemów czasu rzeczywistego;

Obiektowe podejścia do wytwarzania oprogramowania

- System jest analizowany w sposób obiektowy jeśli:
  - Jest dzielony na obiekty posiadające:
    - Tożsamość, Stan, Zachowanie
  - Obiekty są grupowane w klasy złożone z obiektów o podobnych stanach i zachowaniu
- Metody obiektowe:
  - są wygodnym narzędziem analizy złożonych systemów, w szczególności systemów o dużej stronie pasywności i złożonej funkcjonalności
  - ukrywają dane (hermetyzacja)
  - wykorzystują gotowe elementy
  - pozwalają na szybkie prototypowanie i przyrostowy rozwój
  - programowanie oparte na zdarzeniach

Obiektowe podejścia do wytwarzania oprogramowania

- **OODA (Booch Methodology),**
- **Object Modelling Technique - OMT (Rumbaugh),**
- **Objectory(Jacobson),**
- **OOA/OOD(Coad/Yourdon),**
- Express,
- OOSA(Shlaer-Mellor),
- MOSES/OPEN,
- Real-Time Object-Oriented Modelling,
- Schlear-Mellor,
- **UML->RUP;**

Obiektowe podejścia do wytwarzania oprogramowania

- OMT-2 (rozwinięcie OMT-1):
  - technika modelowanie obiektów;
  - nacisk na analizę systemów oprogramowania;
  - słaba w projektowaniu;

- Booch ’93 (powstała z Booch ’91):
  - nacisk na projektowanie i tworzenie systemów oprogramowania;
  - słaba w analizie;

- OODE
  - obiektowa technika programowania;
  - kładła nacisk na modelowanie biznesowe i analizę wymagań;

Obiektowe podejścia do wytwarzania oprogramowania

- Analiza obiektowa
  - opracowanie modelu obiektowego dziedziny zastosowania;
  - rozpoznane obiekty odzwierciedlają byty i operacje związane z rozwiązywanym problemem;
- Projektowanie obiektowe
  - opracowanie modelu obiektowego systemu oprogramowania, który będzie implementacją zidentyfikowanych wymagań;
  - obiekty projektu obiektowego są związane z rozwiązaniem problemu;
- Programowanie obiektowe
  - realizacja projektu oprogramowania za pomocą języka programowania obiektowego;
  - języki obiektowe umożliwiają bezpośrednią implementację obiektów i dostarczają udogodnienia do definiowania klas obiektów;

Faza analizy

- Identyfikacja aktorów:
  - Grupy użytkowników wspierane przez system w:
    - podstawowych i drugoplanowych zadaniach;
    - administrowaniu i utrzymywaniu;
  - Źródła danych wej. i odbiorcy wyników:
    - osoby;
    - zewnętrzne systemy lub urządzenia;
- Identyfikacja przypadków użycia:
  - Jakie zadania dla użytkownika realizuje system?
  - Jakie dane z systemu interesują użytkownika lub system zewnętrzny?
  - Czy są zainteresowani zdarzeniami w systemie?
  - Max liczba przypadków użycia: 5-15, 15-40, 40-100;

Faza analizy

- Identyfikacja klas obiektów - typowe klasy:
  - przedmioty namacalne (np. samochód, czujnik),
  - role pełnione przez osoby (np. pracownik, wykładowca, student),
  - zdarzenia, o których system przechowuje informacje (lądowanie samolotu, wysłanie zamówienia, dostawa),
  - interakcje pomiędzy osobami i/lub systemami o których system przechowuje informacje (np. pożyczka, spotkanie, sesja),
  - lokalizacje - miejsce przeznaczone dla ludzi lub przedmiotów,
  - grupy przedmiotów namacalnych (np. kartoteka, samochód jako zestaw części),
  - organizacje (np. firma, wydział, związek),
  - wydarzenia (np. posiedzenie sejmu, demonstracja uliczna),
  - koncepcje i pojęcia (np. zadanie, miara jakości),
  - dokumenty (np. faktura, prawo jazdy),
  - interfejsy do systemów zewnętrznych,
  - interfejsy do urządzeń sprzętowych;

Faza analizy

- Obiekty, zbiory obiektów i metadane:
  - W wielu przypadkach przy definicji klasy należy dokładnie ustalić, z jakiego rodzaju abstrakcją obiektu mamy do czynienia;
  - Należy zwrócić uwagę na następujące aspekty:
    - czy mamy do czynienia z konkretnym obiektem w danej chwili czasowej?
    - czy mamy do czynienia z konkretnym obiektem w pewnym odcinku czasu?
    - czy mamy do czynienia z opisem tego obiektu (dokument, metadane)?
    - czy mamy do czynienia z pewnym zbiorem konkretnych obiektów?
- Np. klasa „samochód” - może chodzić o:
  - egzemplarz samochodu wyprodukowany przez pewną fabrykę;
  - model samochodu produkowany przez fabrykę;
  - pozycję w katalogu samochodów opisujący własności modelu;
  - historię stanów pewnego konkretnego samochodu;

Faza analizy

- Zalecenia dotyczące identyfikacji klas:
  - Wyraźnie zdefiniować kontekst (w tym opis) klasy;
  - Unikać w nazwie synonimów i nazw zbliżonych znaczeniowo;
  - Pomijać klasy, które nie mieszczą się w zakresie systemu;
  - Wyeliminować klasy będące w rzeczywistości:
    - atrybutami lub grupami atrybutów (właściwościami) innych klas;
    - operacjami (usługami) innych klas;
    - związkami lub rolami pełnionymi w związkach przez inne klasy;
    - → można połączyć takie byty w większe klasy;
  - Utworzyć wiele klas z jednej, jeżeli grupuje atrybuty lub operacje kontekstowo odległe;
  - Uzupełnić o atrybuty opisujące klasy w kontekście systemu;
  - Klasy kontekstowo powiązane pogrupować w podsystemy – np. kandydatami są grupy powiązane silnymi relacjami (np. dziedziczenie);
  - Unikać w tej fazie klas reprezentujących elementy implementacyjne;

Faza analizy

- Zalecenia dotyczące identyfikacji związków klas:
  - Unikać związków bez klasy docelowej;
  - Pomijać związki z elementami spoza systemu;
  - Dążyć do związków dwuelementowych;
  - Zwracać szczególną uwagę na związki trwałe w czasie – związki nietrwałe wykorzystać podczas konstrukcji modelu dynamicznego (np. do budowy komunikatów);
  - Kompletować związki i role klas w związkach;
  - Uszczegółowić związki o krotności obiektów;
  - Klasy o podobnych cechach powiązać relacją dziedziczenia;
  - Zweryfikować dostępność informacji w modelu klasy-związki-klasy z różnych punktów startowych;
  - Unikać w tej fazie związków reprezentujących zależności implementacyjne;

Faza analizy

- Zalecenia dotyczące modelu dynamicznego klas:
  - Koncentrować się na behawioralnych aspektach systemu;
  - Rozpatrzeć interakcje związane z pożądanym, błędnym i awaryjnym zachowaniem systemu;
  - Kompletować „trójki”: Nadawca (Aktor lub obiekt)-zdarzenie-Odbiorca;
  - Przedstawić uporządkowany w czasie przepływ zdarzeń w systemie dla każdej klasy;
  - Zdarzenia odpowiadające jednej klasie należy łączyć we wspólny diagram;

Faza analizy

- Kluczowe czynniki sukcesu fazy analizy
  - Zaangażowanie właściwych osób ze strony klienta;
  - Kompleksowe i całościowe podejście do problemu, nie koncentrowanie się na partykularnych jego aspektach;
  - Nie unikanie trudnych problemów (bezpieczeństwo, skalowalność, szacunki objętości i przyrostu danych, itd.);
  - Zachowanie przyjętych standardów, np. w zakresie notacji;
  - Sprawdzenie poprawności i wzajemnej spójności modelu;
  - Przejrzystość, logiczny układ i spójność dokumentacji;

Od analizy do szczegółowego projektu obiektów

- Celem projektowania jest opracowanie szczegółowego opisu implementacji systemu.
- W odróżnieniu od analizy, w projektowaniu dużą rolę odgrywa środowisko implementacji.
- Dwa etapy fazy projektowania:
  - projekt strategiczny, projekt systemy (strategic, system design):
    - podział na podsystemy,
    - współbieżność,
    - przechowywanie danych,
    - sterowanie.
  - projekt szczegółowy (detailed design):
    - uściślanie definicji klas,
    - dziedziczenie,
    - optymalizacja modelu,
    - modularyzacja,
    - ukrywanie informacji;

Faza projektowania

- Zadania w etapach fazy projektowania:
  - uściślenie istniejących definicji klas, np. metod,
  - dziedziczenie klas i operacji,
  - szczegółowy projekt operacji wraz z przeprojektowaniem ich algorytmów,
  - wprowadzenie ogólnych mechanizmów realizacji dynamiki obiektów,
  - decyzje o trwałości obiektów,
  - modularyzacja i ukrywanie informacji,
  - optymalizacja modelu,
  - dokumentacja projektu;
- Zatem projektanci muszą więc posiadać dobrą znajomość języków, bibliotek, i narzędzi stosowanych w trakcie implementacji;

Faza projektowania

- Zadania fazy projektowania – przykład uściślenia definicji metod;
  - Podanie nagłówków metod oraz ich parametrów.
  - Określenie, które z metod będą realizowane jako funkcje wirtualne (poźno wiązane) a które jako zwyczajne funkcje (wiązane statyczne).
  - Zastąpienie niektórych prostych metod bezpośrednim dostępem do atrybutów.
    - Np. metody `PobierzNazwisko, UstawNazwisko`, etc.
  - Zastąpienie niektórych atrybutów redundantnych przez odpowiednie metody, np.
    - `Wiek = BieżącaData - DataUrodzenia;`
    - `KwotaDochodu = KwotaPrzychodu - KwotaKosztów;`

Faza projektowania

- Zadania fazy projektowania – przykład sposobu implementacji związków (asocjacji);
  - Związki można zaimplementować na wiele sposobów, z reguły poprzez wprowadzenie dodatkowych atrybutów (pól) - mogą one być następujące:
    - obiekty powiązanej klasy,
    - wskaźniki (referencje) do obiektów powiązanej klasy,
    - identyfikatory obiektów powiązanej klasy,
    - klucze kandydujące obiektów powiązanej klasy;
  - W zależności od przyjętego sposobu oraz od liczności związków (*1:1, 1:n, n:1, m:n*) możliwe są bardzo różne deklaracje w przyjętym języku programowania.

Tablica obiektów: ⟶ `TypSłowoKluczowe SłowaKluczowe[100];`
Lista wskaźników: ⟶ `list< TypSłowoKluczowe *> SłowaKluczowe;`
Tablica wskaźników: ⟶ `char * WskaźnikiSłówKluczowych[100];`

Podstawowe rezultaty fazy projektowania

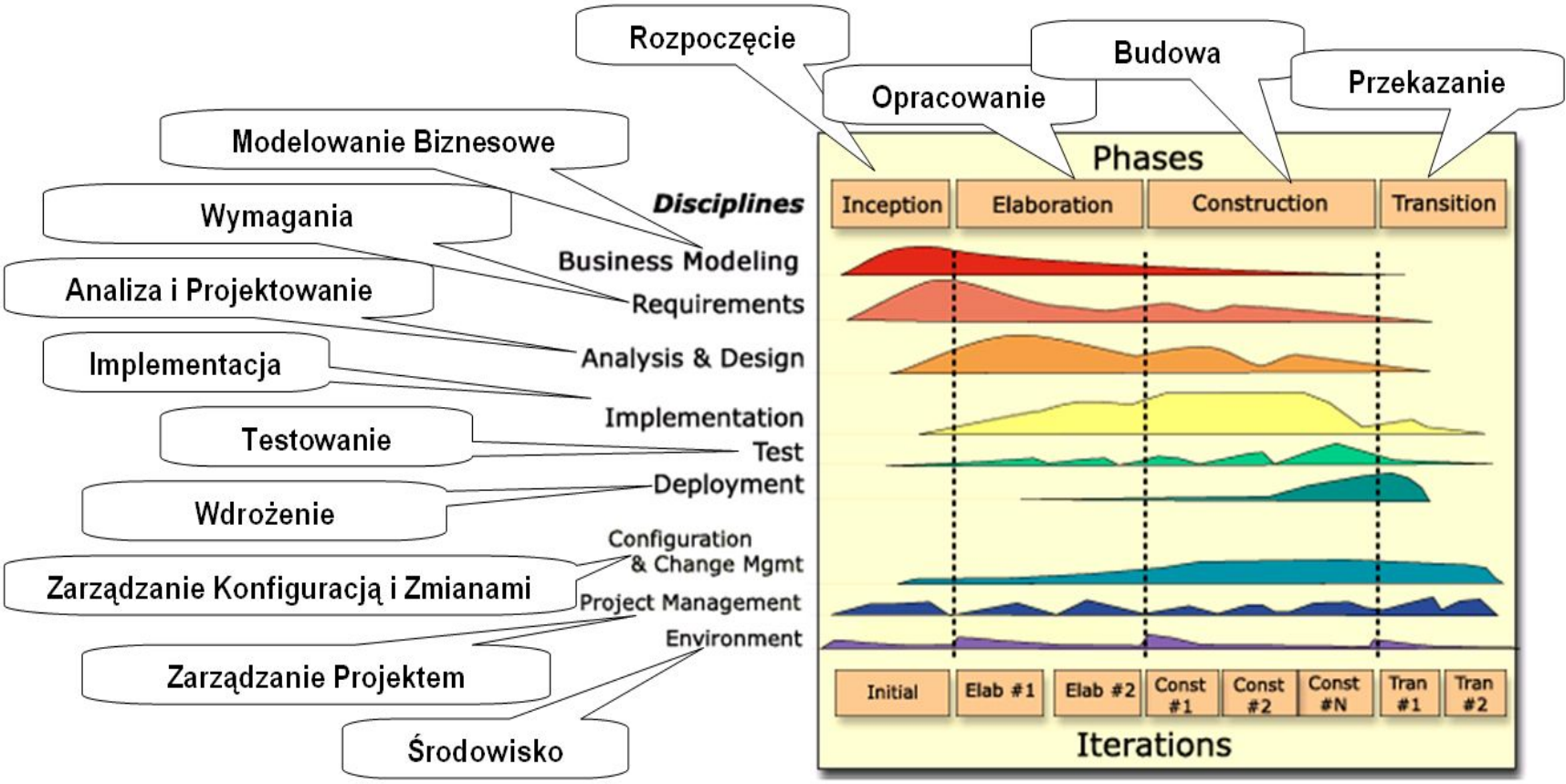
- Poprawiony i uszczegółowiony dokument opisujący wymagania;
- Poprawione i uszczegółowione modele;
- Uszczegółowiona specyfikacja słownika danych;
- Dokument opisujący stworzony projekt składający się z (dla obiektowych):
  - diagramu klas
  - diagramów interakcji obiektów
  - diagramów stanów
  - innych diagramów, np. diagramów modułów, konfiguracji
  - zestawień zawierających:
    - definicje klas
    - definicje atrybutów
    - definicje danych złożonych i elementarnych
    - definicje metod
- Zasoby interfejsu użytkownika, np. menu, dialogi;
- Projekt bazy danych;
- Projekt fizycznej struktury systemu;
- Poprawiony plan testów;
- Pierwszy harmonogram implementacji;

RUP

- Ukierunkowany na przypadki użycia
- Architekturo-centryczny
- Iteracyjny
- Przyrostowy
- Sterowany ryzykiem

RUP

- Dwa wymiary RUP
  - FAZY (ang. *phases*)
  - PRZEPŁYWY, DYSCYPLINY (ang. *disciplines*)



RUP

- Proces budowy systemu informatycznego składa się z dyscyplin, z których każda dzielona jest na fazy:
  - Rozpoczęcie
  - Opracowanie
  - Budowa
  - Przekazanie
- Kamienie milowe
- Podejście iteracyjne

O czym teraz…

- Geneza i charakterystyka UML;
  - Zapoznanie z wybraną notacją wykorzystywaną w modelowaniu, analizie i projektowaniu systemów informatycznych;

UML – notacja obiektowa

- Rodzaje notacji:
  - tekstowo-opisowa,
  - specyfikacje - ustrukturalizowany zapis tekstowy i numeryczny,
  - notacje graficzne;
- Jeżeli notacja posiada składnię (np. symbole graficzne) oraz semantykę (znaczenie symboli graficznych), staje się językiem;
- Szczególną uwagę skupimy na notacji graficznej;
- Omówimy notację (język) *UML*….

UML – notacja obiektowa

OOSE
Ivar Jacobson

UNIFIED MODELING LANGUAGE

OMT
Jim Rumbaugh

Unified Method
1995 v0.8

UML
1996 v0.9

UML
1997 v1.0

OMG
UML Standard

...

UML
1999 v1.3

...

UML
2004 v2.0

OOADA
Grady Booch

Partnerzy
(IBM, Microsoft,
HP, DEC, ...)

OMG Revision
Task Force

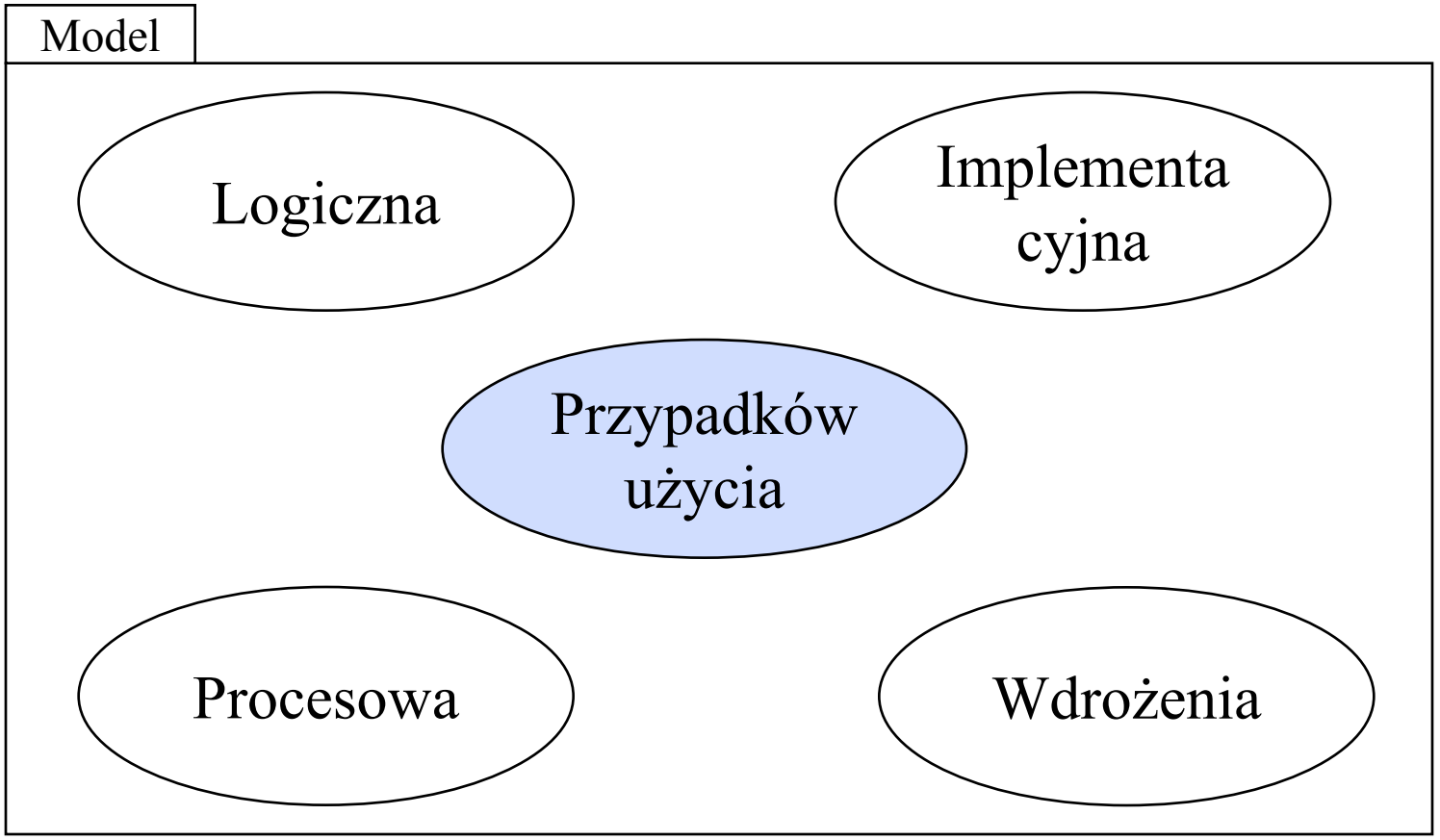
Fragmentacja → Unifikacja → Standaryzacja → Industrializacja

UML – notacja obiektowa

- Unified Modeling Language – UML
  - The Unified Modeling Language™ (UML™) is the industry-standard language for specifying, visualizing, constructing, and documenting the artifacts of software systems. *(http://www-306.ibm.com/software/rational/uml/)*
  - Znormalizowany język graficzny służący do specyfikowania, tworzenia i dokumentowania wyników (np. modeli) systemów informatycznych;
  - Cechy:
    - *uniwersalny* – może być stosowany do modelowania zarówno systemów informacyjnych, systemów WWW, systemów czasu rzeczywistego;
    - wspomaga jednoznacznie i szczegółowo *specyfikowanie* istotnych decyzji etapów analizy, projektu i implementacji;
    - umożliwia dokumentowanie architektury systemu i wszystkich jego szczegółów w postaci tzw. *artefaktów*: wymagania, architektura, projekt, kod źródłowy, plany projektu, testy, prototypy, kolejne wersje.

UML – składowe

- Perspektywy modelowe – 4+1:



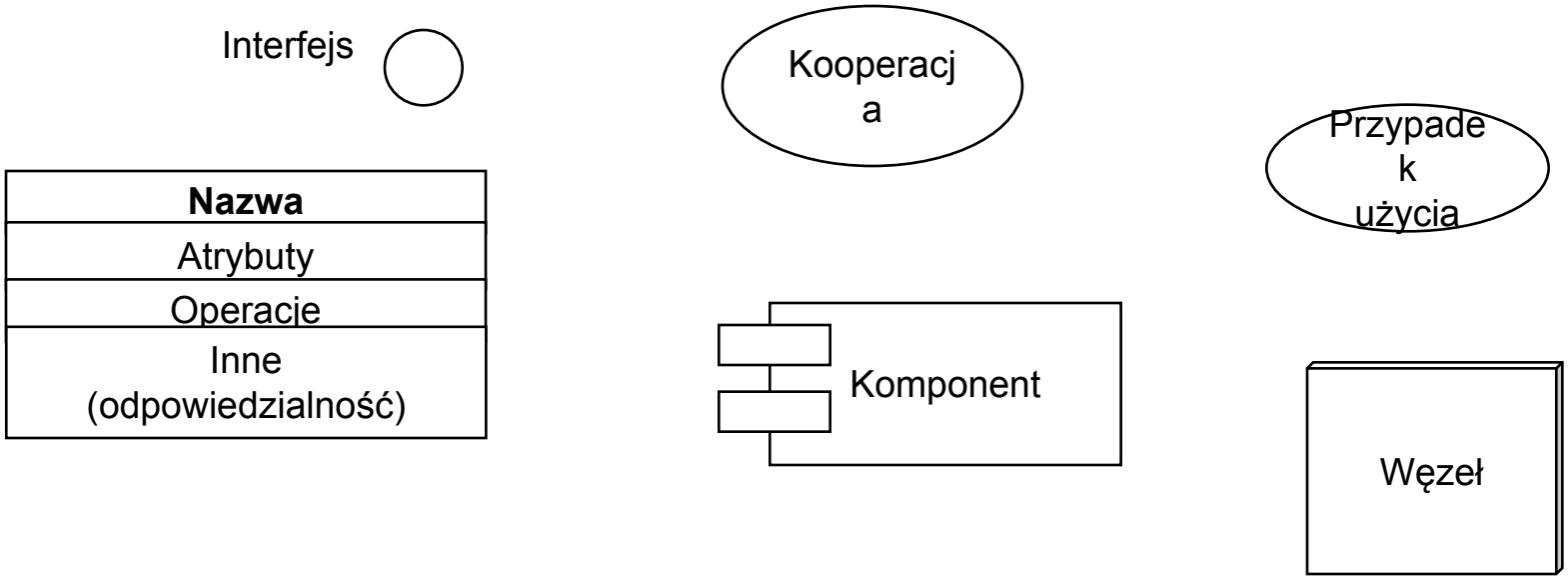
UML – składowe

- Słownik UML dzieli się na trzy grupy:
  - elementy,
  - związki (relacje),
  - diagramy;

- *Model* – kolekcja diagramów i informacji dodatkowych, opisujących statyczne i dynamiczne aspekty modelowanego systemu;

UML – elementy

- Elementami UML są podstawowe obiektowe bloki konstrukcyjne stosowane do budowy modeli:
  - *strukturalne*
    - statyczne części modelu reprezentujące składniki pojęciowe lub fizyczne;
    - *klasa, interfejs, przypadek użycia, komponent, węzeł, kooperacja (grupa współdziałania)*;

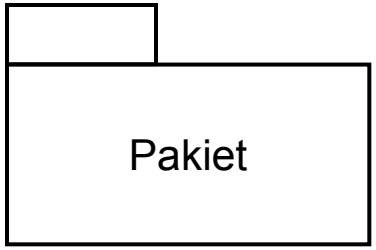


UML – elementy

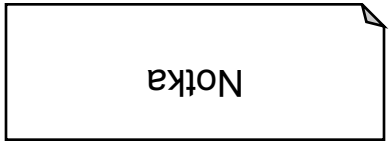
- Elementami UML są podstawowe obiektowe bloki konstrukcyjne stosowane do budowy modeli:
  - *czynnościowe*
    - elementy dynamiczne wyrażone czasownikami
    - *interakcja, stan*;

UML – elementy

- Elementami UML są podstawowe obiektowe bloki konstrukcyjne stosowane do budowy modeli:
  - *grupujące*
    - bloki organizacyjne modelu;
    - *pakiet*;



  - *komentujące*
    - elementy objaśniające dla uwypuklenia lub zaznaczenia składników;
    - *notatka*;

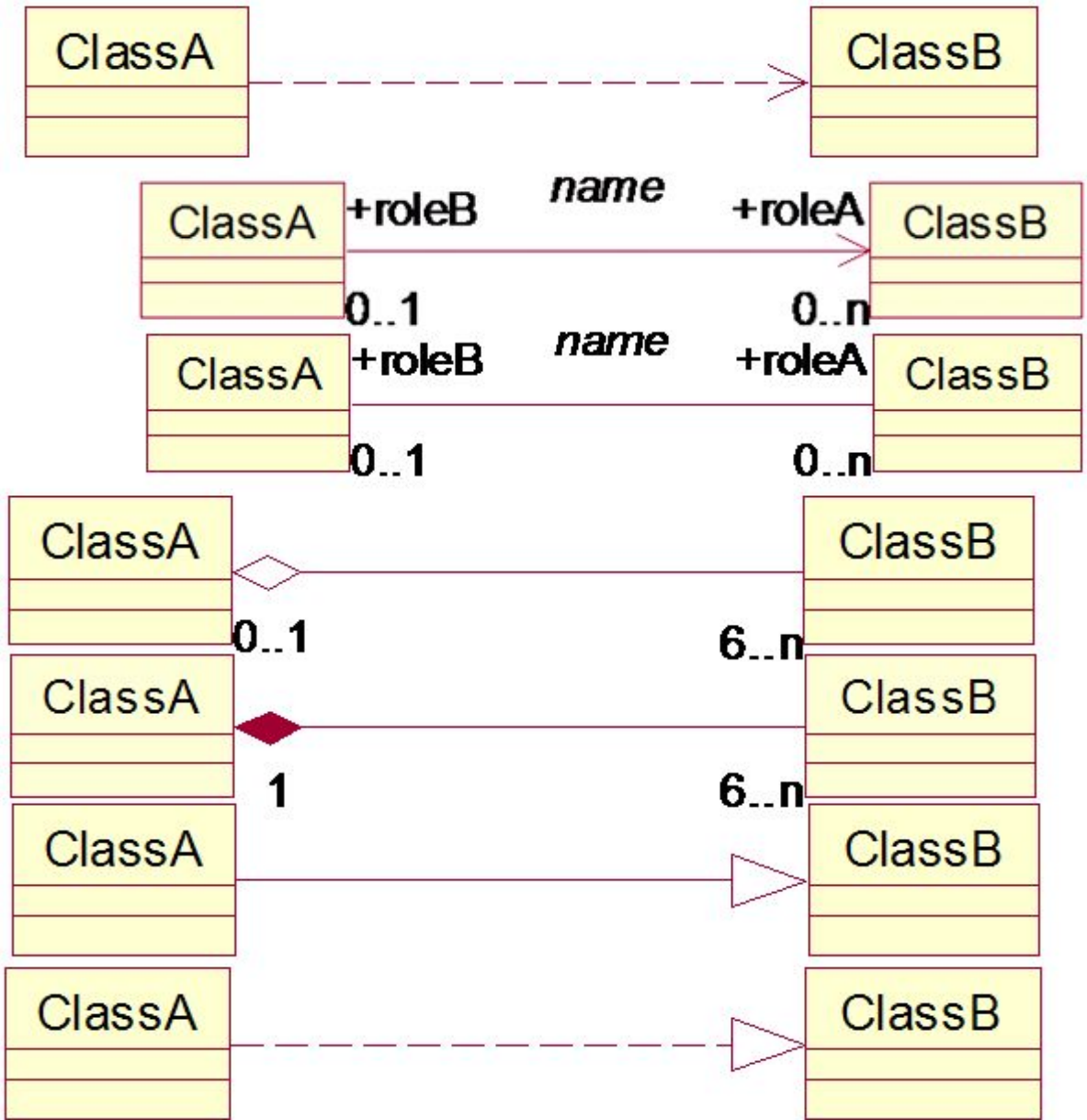


UML – związki

- Związki:
  - służą do łączenia elementów;
  - w praktyce najczęściej stosowane są powiązanie i uogólnienie;
  - *zależność, powiązanie (asocjacja), uogólnienie (generalizacja), realizację;*

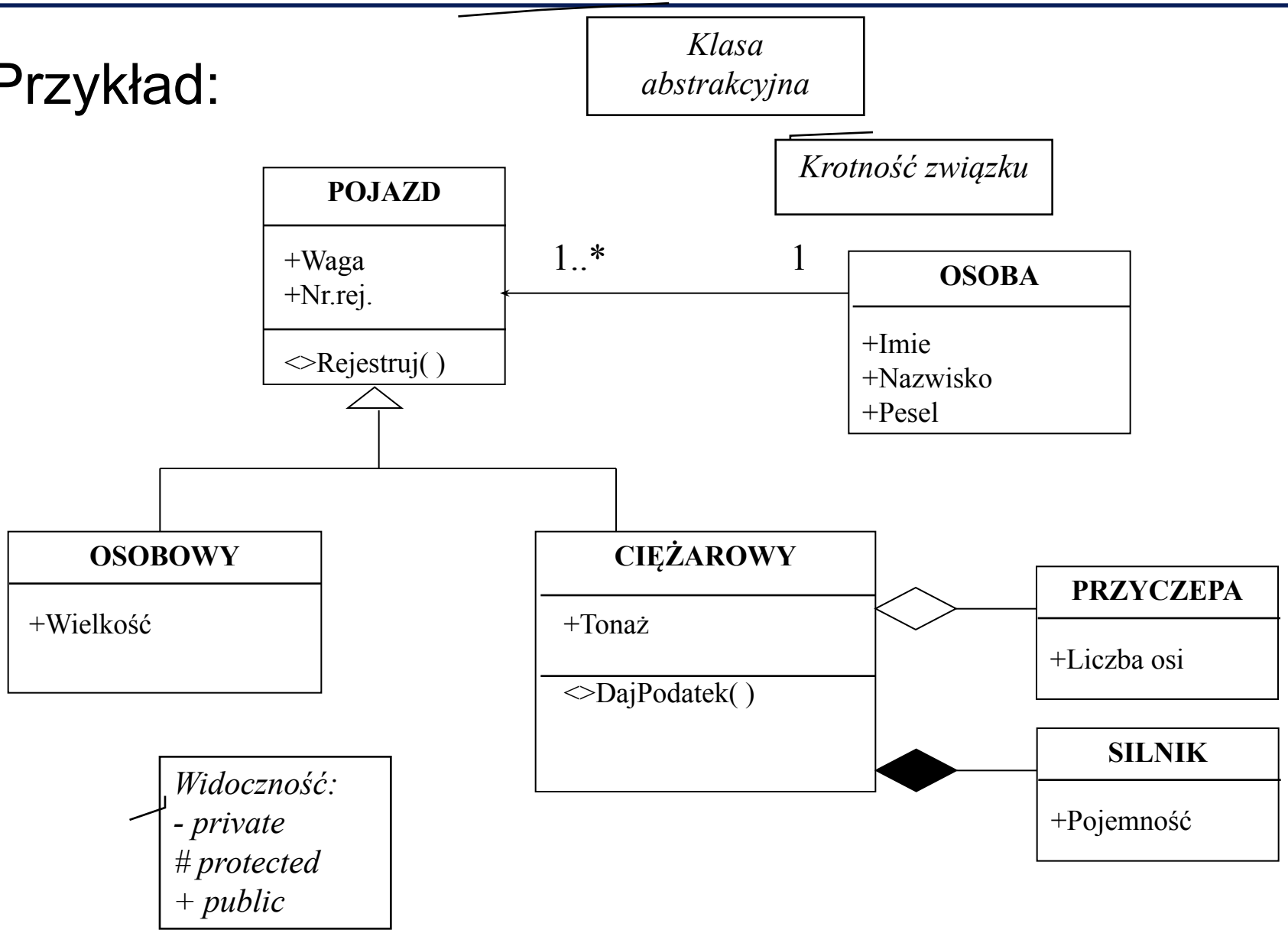
UML – związki

- Związki klas:
  - Zależność
  - Asocjacja
    - Jednokierunkowa
    - Dwukierunkowa
  - Agregacja
  - Kompozycja
  - Generalizacji
  - Realizacja



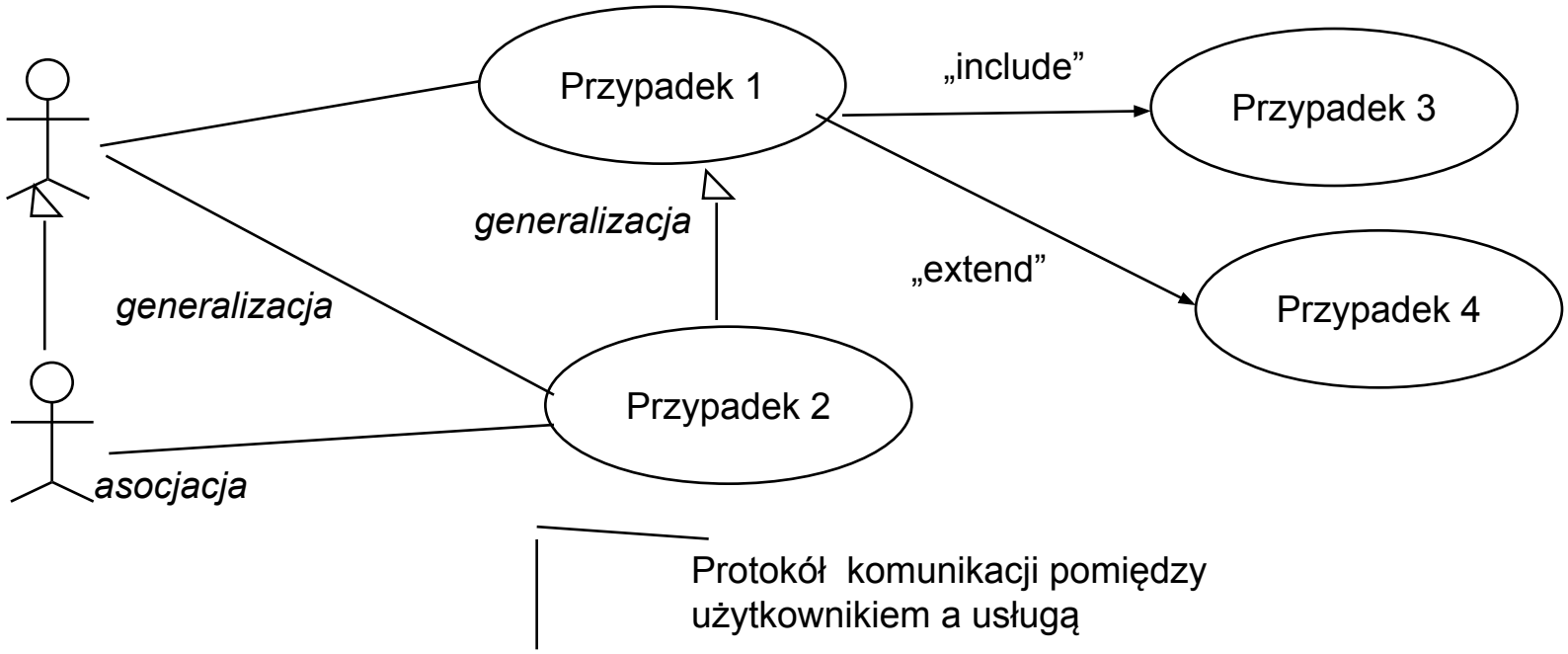
UML – notacja związków

- Przykład:



UML – notacja diagramów

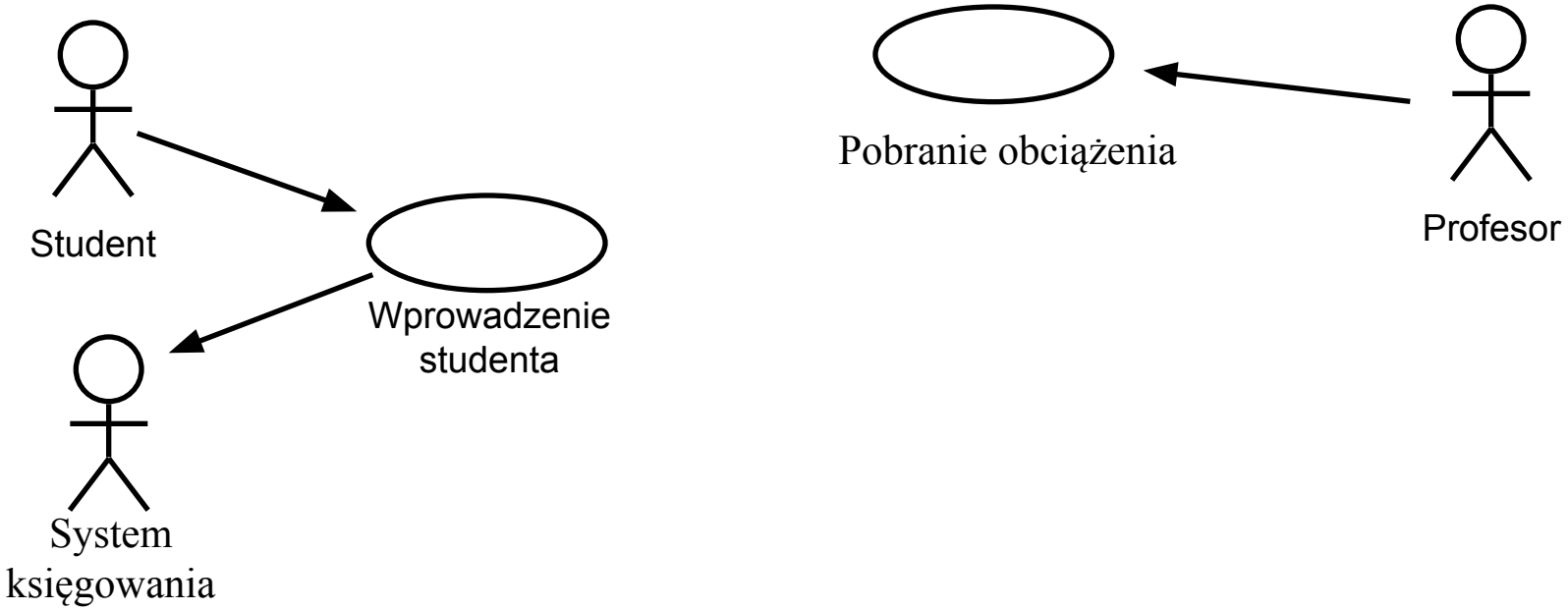
- Diagram przypadków użycia:



„include” oraz „extend” są standardowymi stereotypami uszczegóławiającymi związek

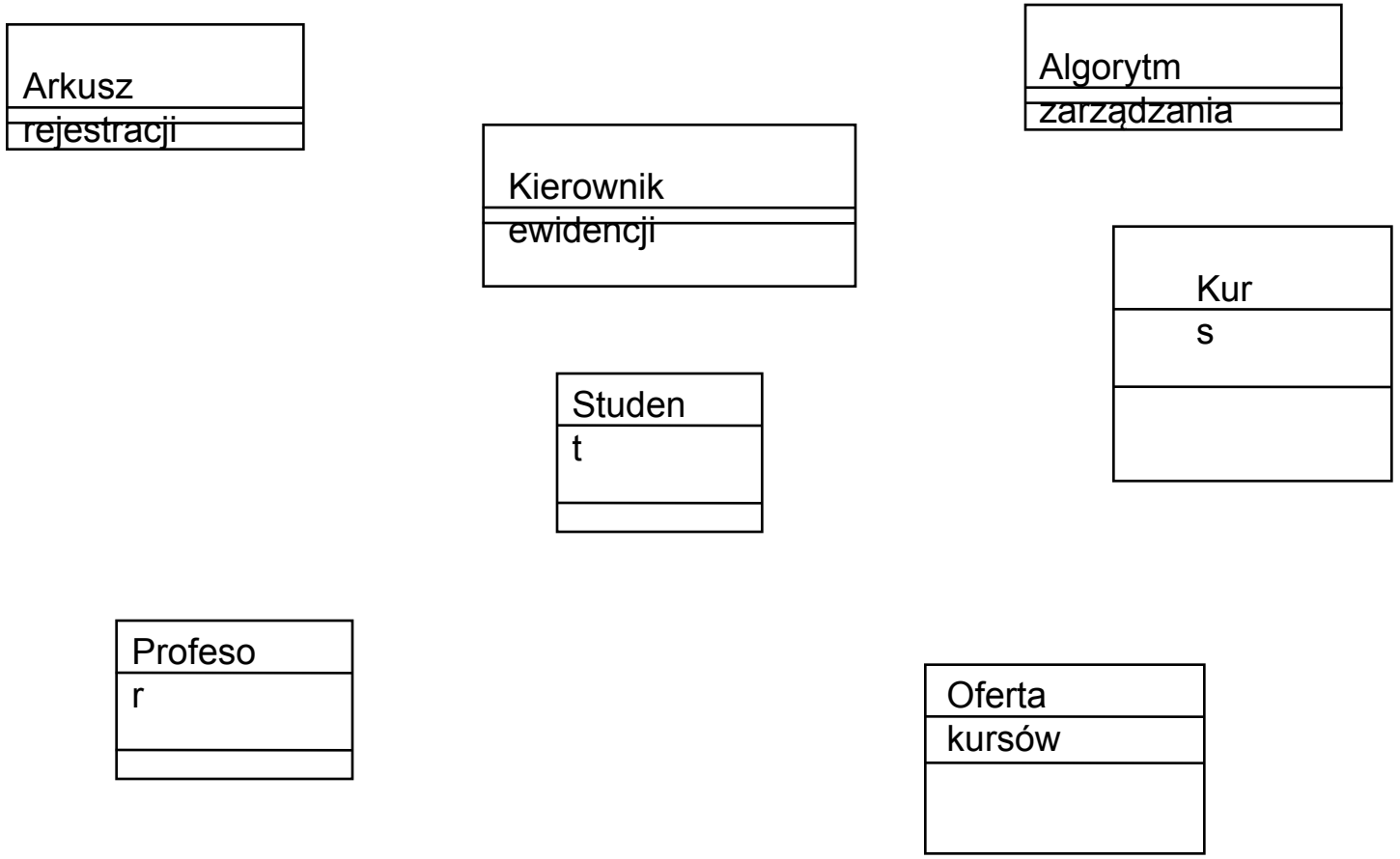
UML – przykład systemu ewidencji studentów

- Diagram przypadków użycia



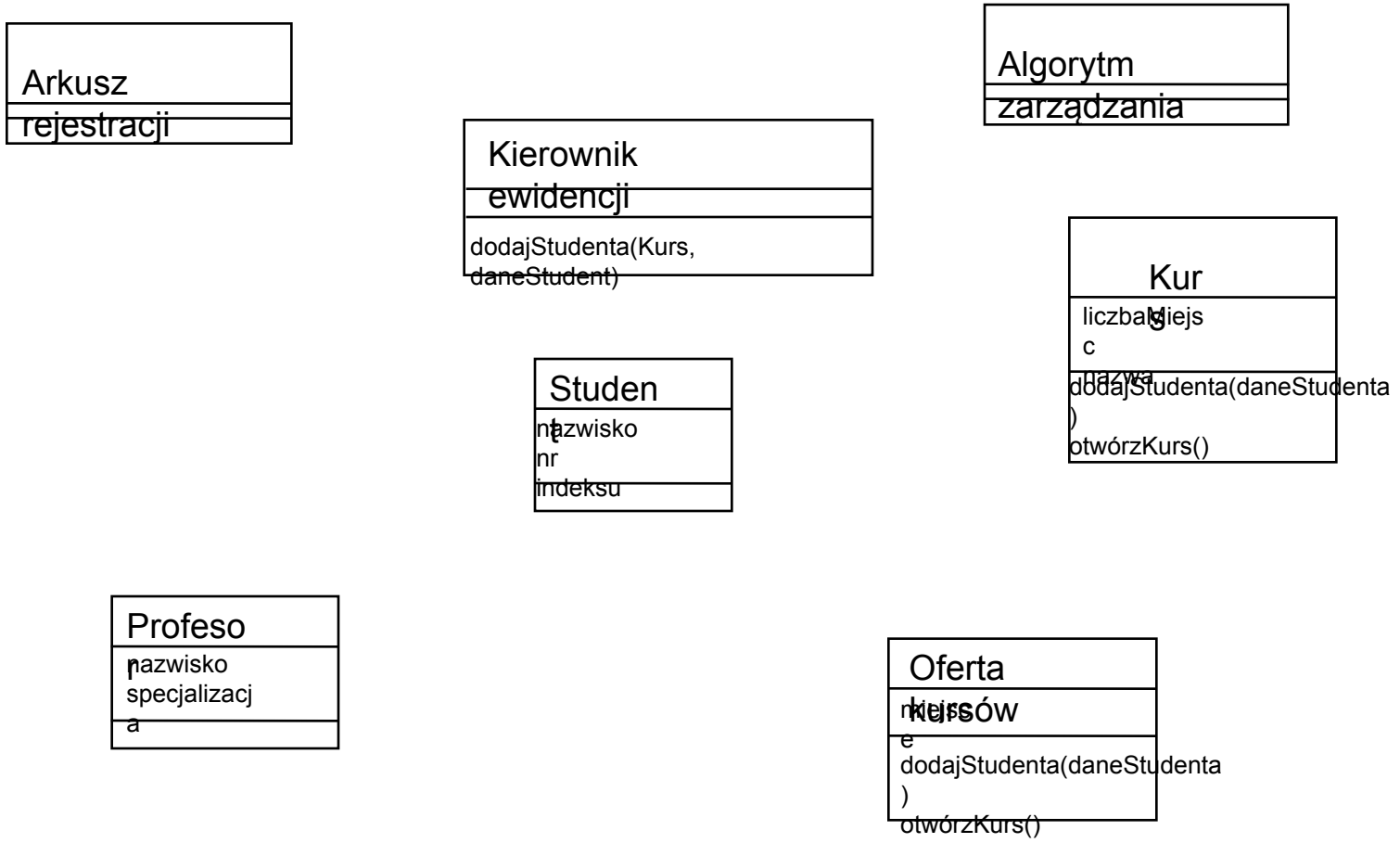
UML – przykład systemu ewidencji studentów

- Diagram klas (1) – klasy abstrakcyjne



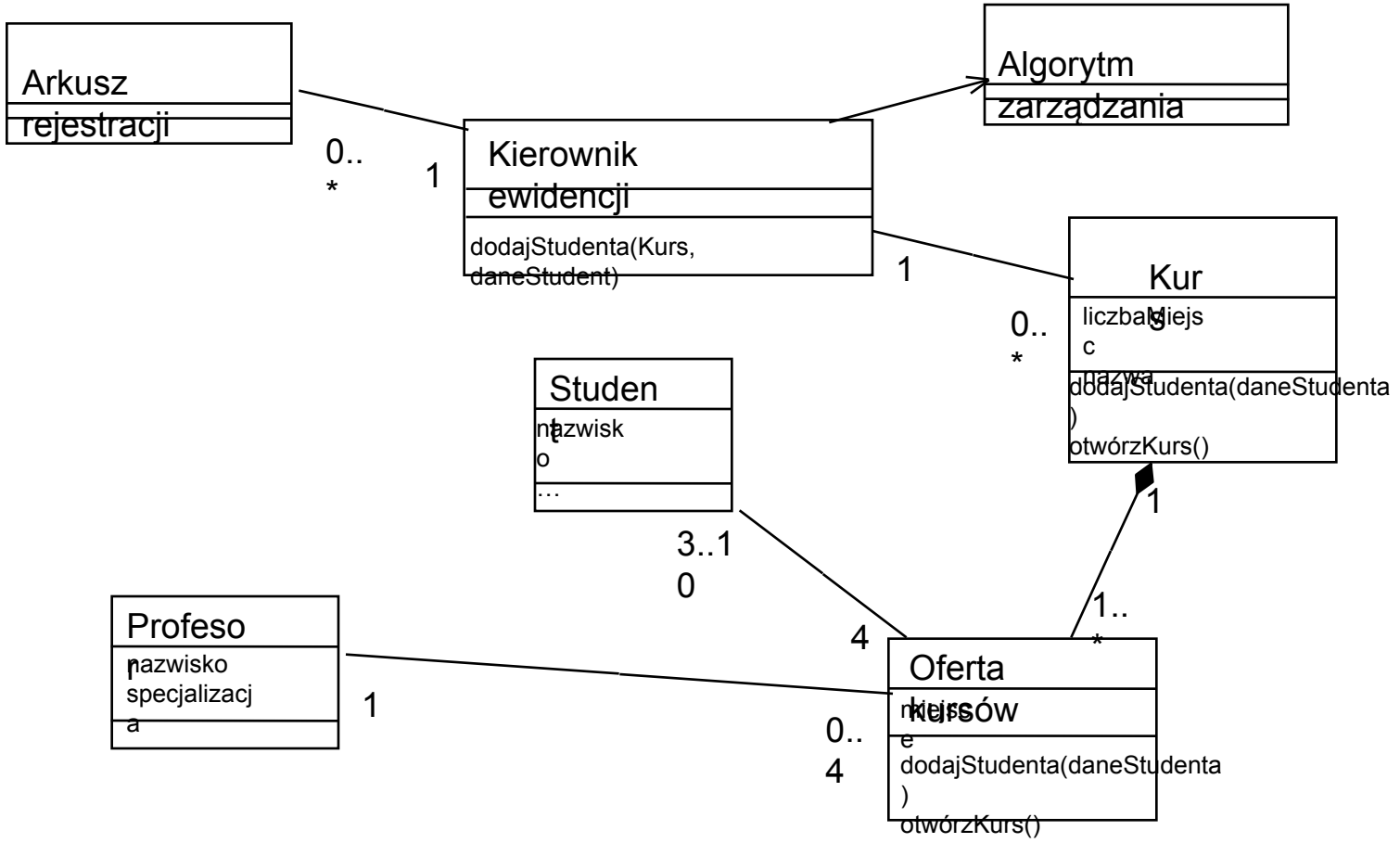
UML – przykład systemu ewidencji studentów

- Diagram klas (2) – klasy uszczegółowione



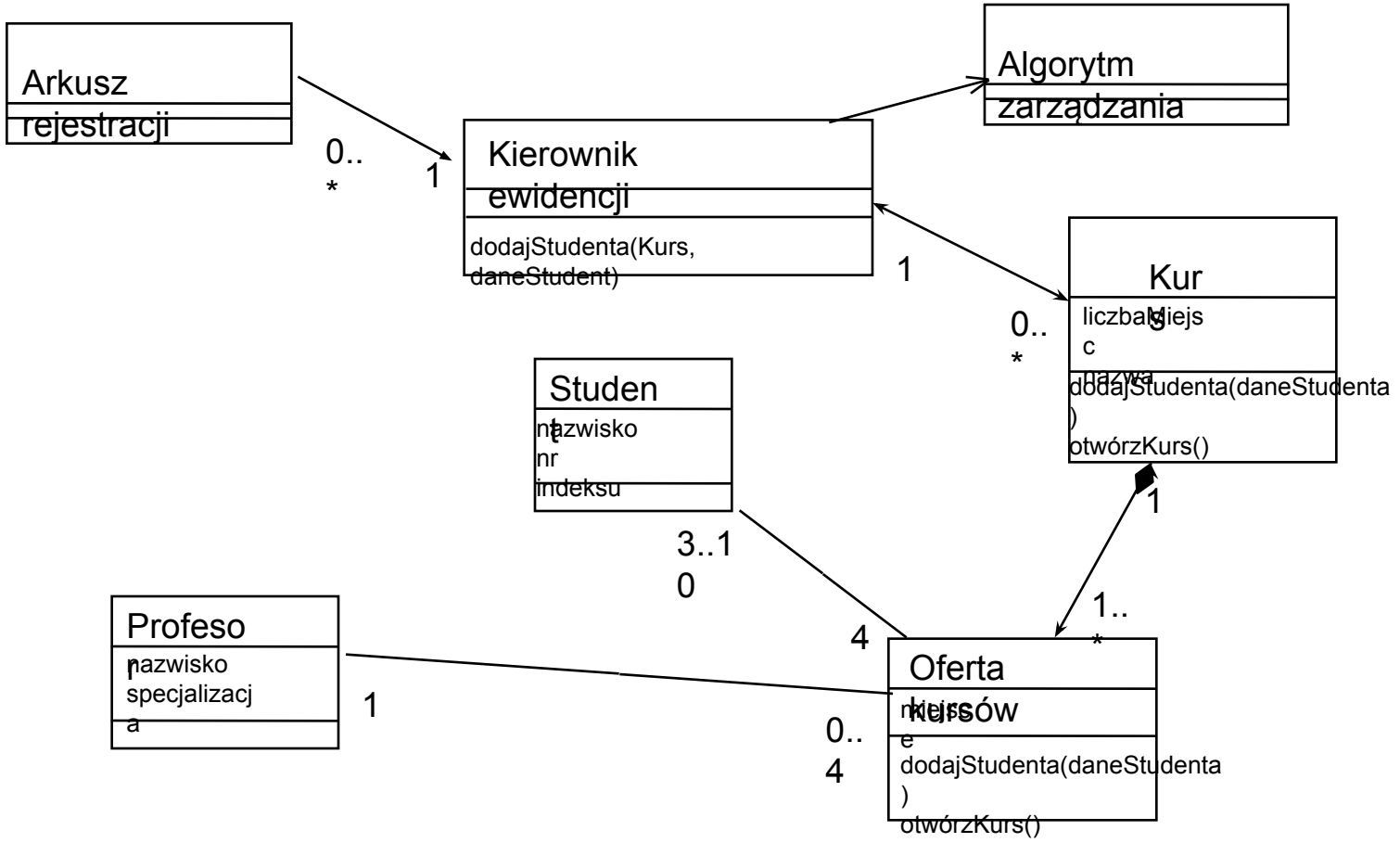
UML – przykład systemu ewidencji studentów

- Diagram klas (3) – związki klas



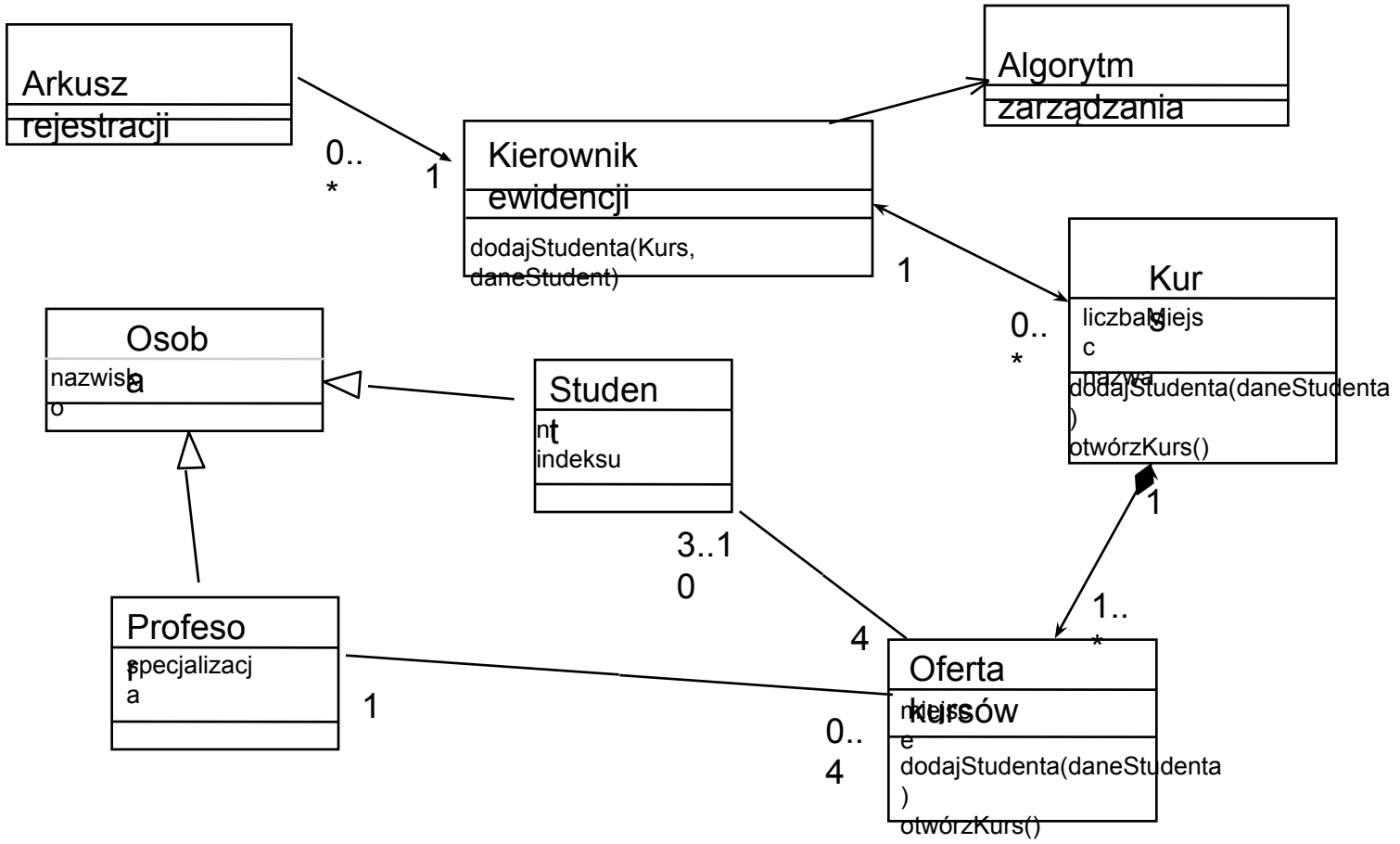
UML – przykład systemu ewidencji studentów

- Diagram klas (4) – skierowanie i krotności związków



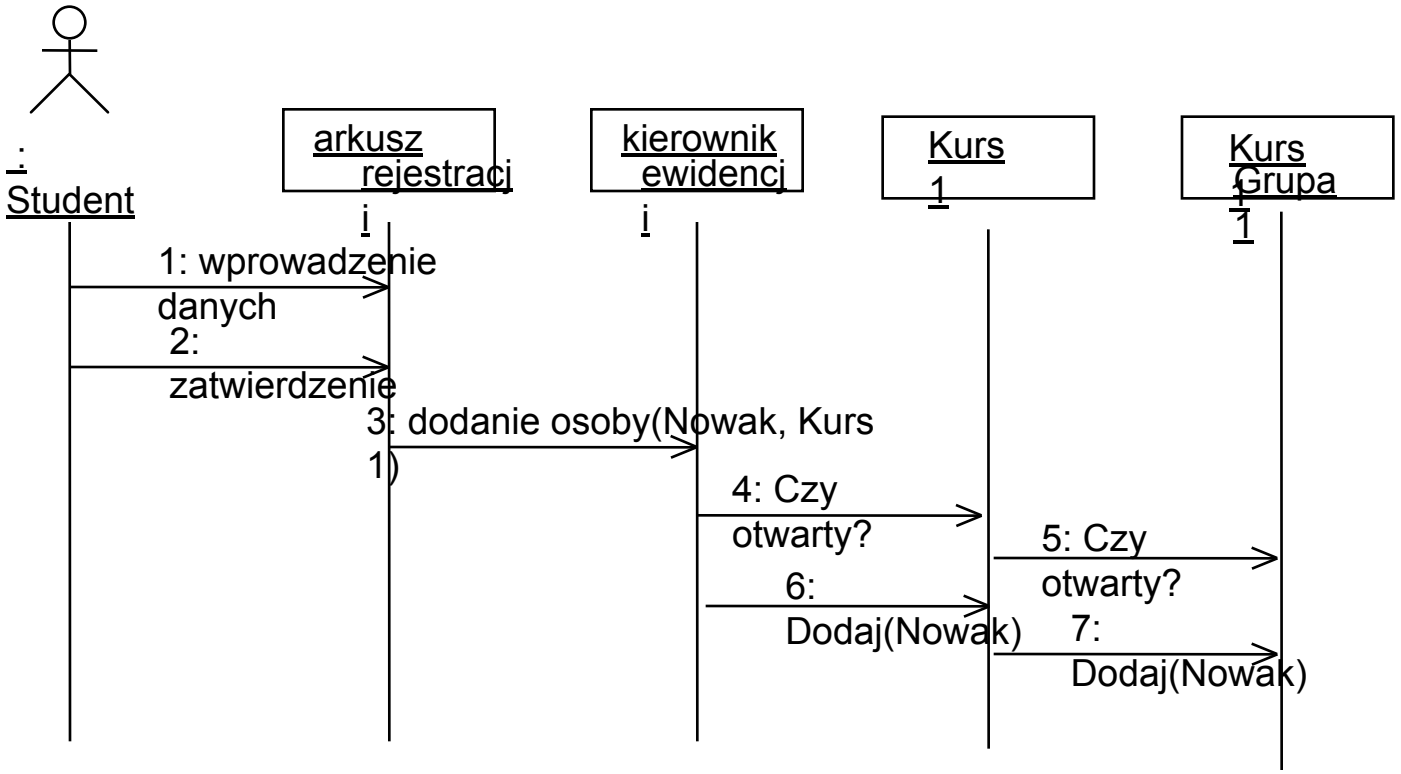
UML – przykład systemu ewidencji studentów

- Diagram klas (5) – generalizacja



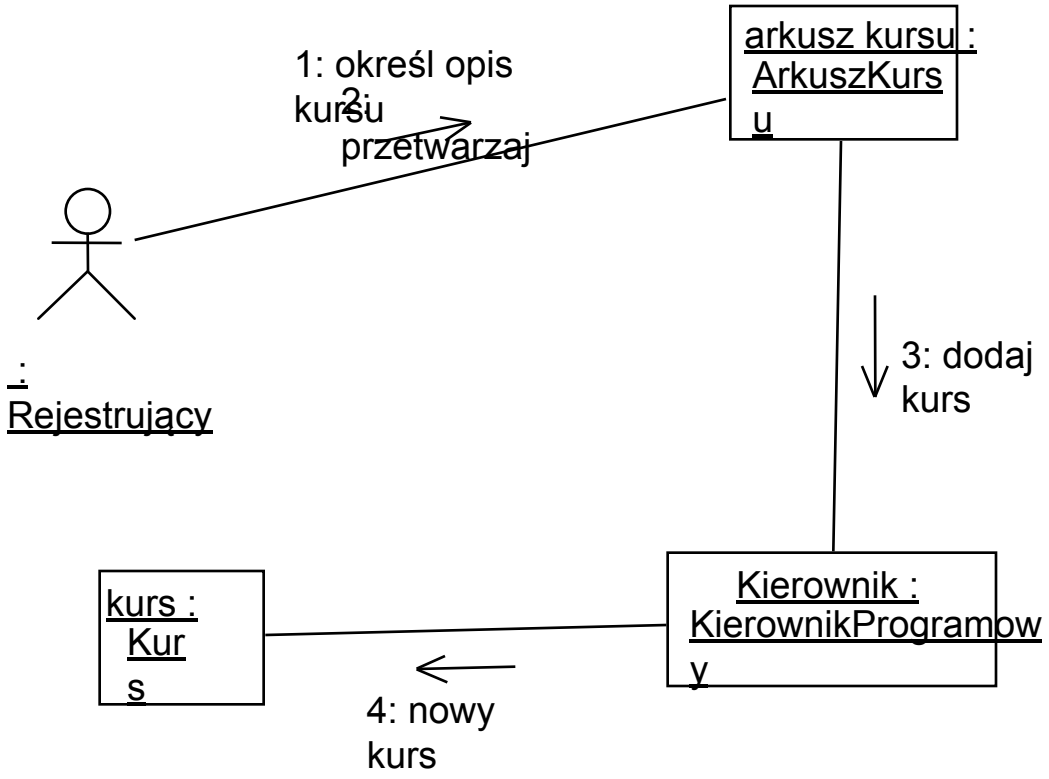
UML – przykład systemu ewidencji studentów

- Diagram sekwencji zdarzeń



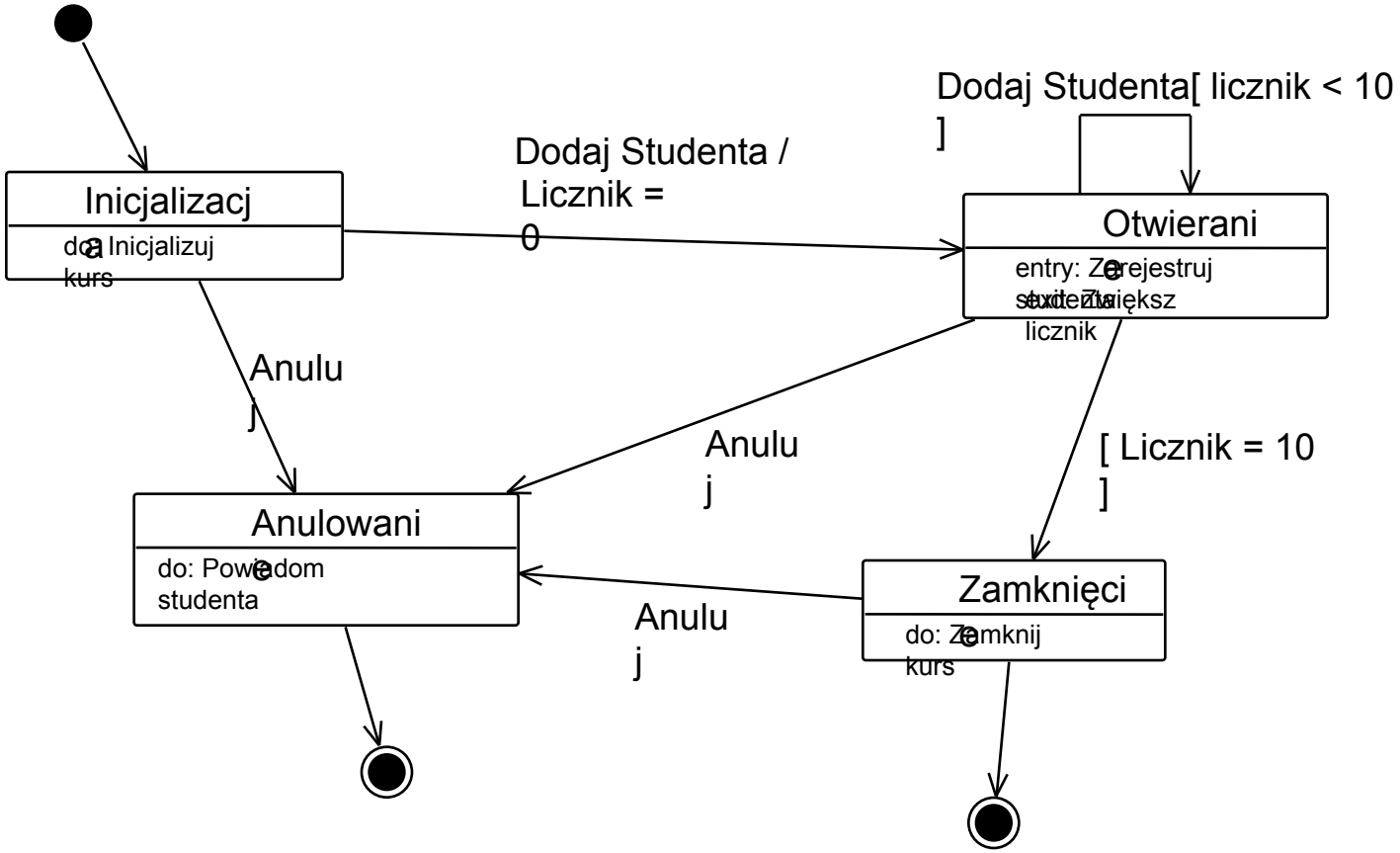
UML – przykład systemu ewidencji studentów

- Diagram współpracy



UML – przykład systemu ewidencji studentów

- Diagram stanów



System mapy pogody

Przykład z książki Iana Sommerville'a „Inżynieria oprogramowania"

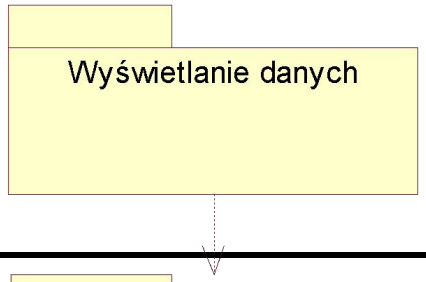
- System tworzący mapy pogody ma regularnie generować mapy pogody;
- Źródła danych:
  - zdalne, niestrzeżone stacje meteorologiczne,
  - inne źródła danych: obserwatorzy pogody, balony i satelity;
- Stacje meteorologiczne
  - przekazują dane do komputera obszaru na jego żądania;
- System komputera obszaru
  - weryfikuje i integruje dane zebrane z różnych źródeł;
- Po zintegrowaniu dane są archiwizowane w zbiorach;
- Lokalne mapy pogody są tworzone na podstawie archiwum i bazy danych map komputerowych;
- Mapy można drukować lub wyświetlać w różnych formatach;

System mapy pogody

- Zadania systemu:
  - zbieranie danych,
  - integracja danych z różnych źródeł,
  - archiwizowanie danych,
  - tworzenie map pogody;
- Każdy etap działania zależy jedynie od wyników przetwarzania z poprzedniego etapu;
- Proponowana architektura:
  - warstwowa,
  - odzwierciedla etapy przetwarzania danych w systemie: zbieranie danych, integrację danych itd.;

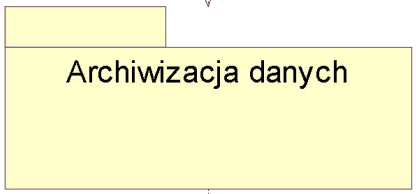
Architektura warstwowa systemu mapy pogody

Propozycja architektury systemu



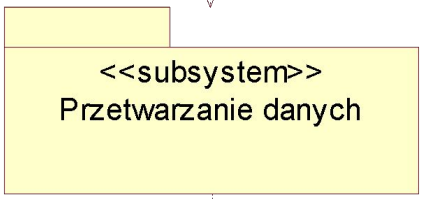
Warstwa wyświetlania danych

Obiekty warstwy przygotowują dane w postaci zrozumiałej dla człowieka



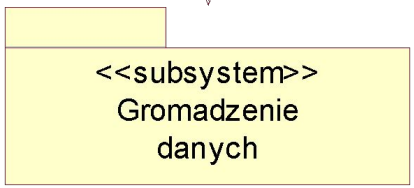
Warstwa archiwizacji danych

Obiekty warstwy zajmują się składowaniem danych



Warstwa przetwarzania danych

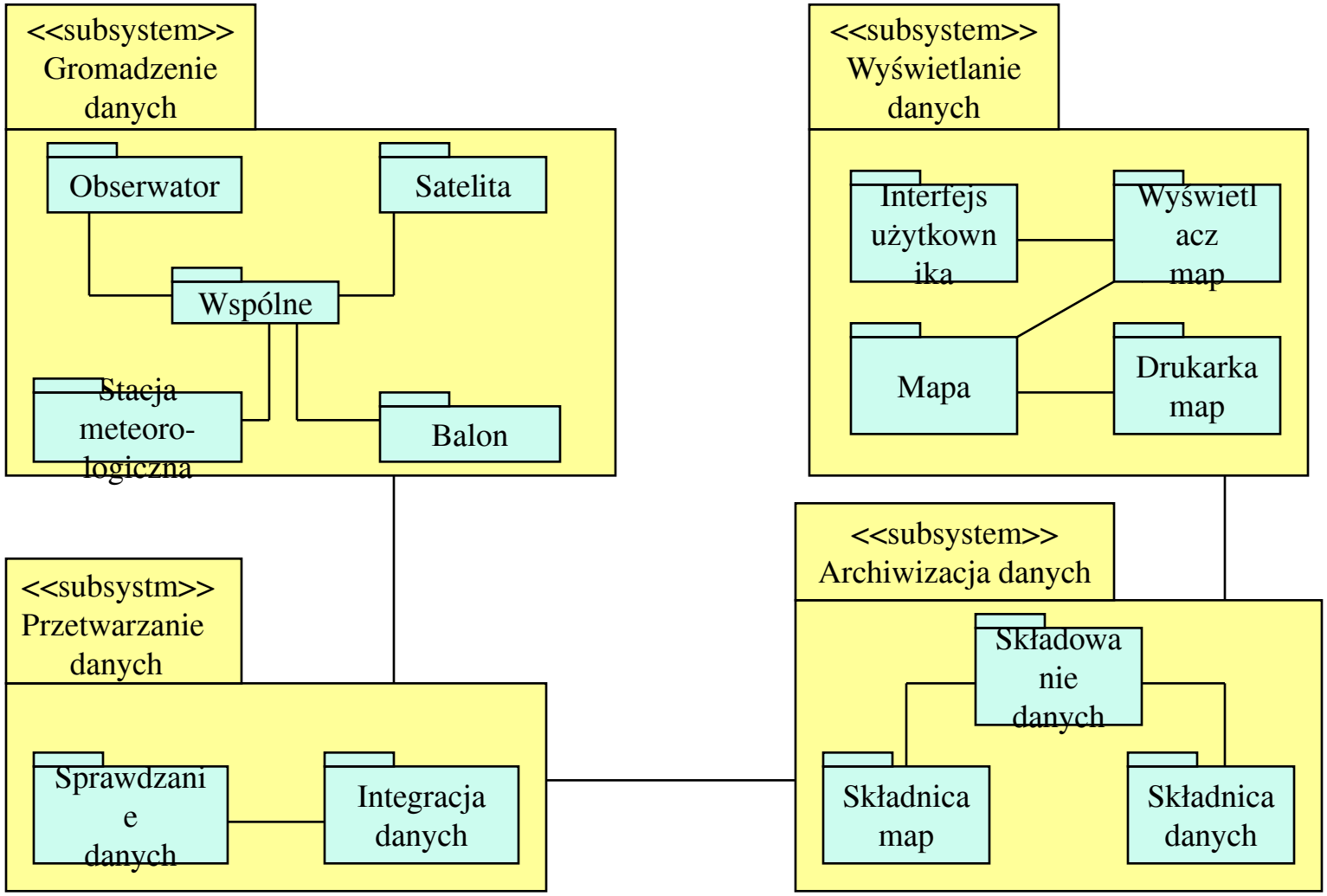
Obiekty warstwy weryfikują i integrują dane



Warstwa gromadzenia danych

Obiekty warstwy zajmują się pozyskaniem danych ze zdalnych źródeł

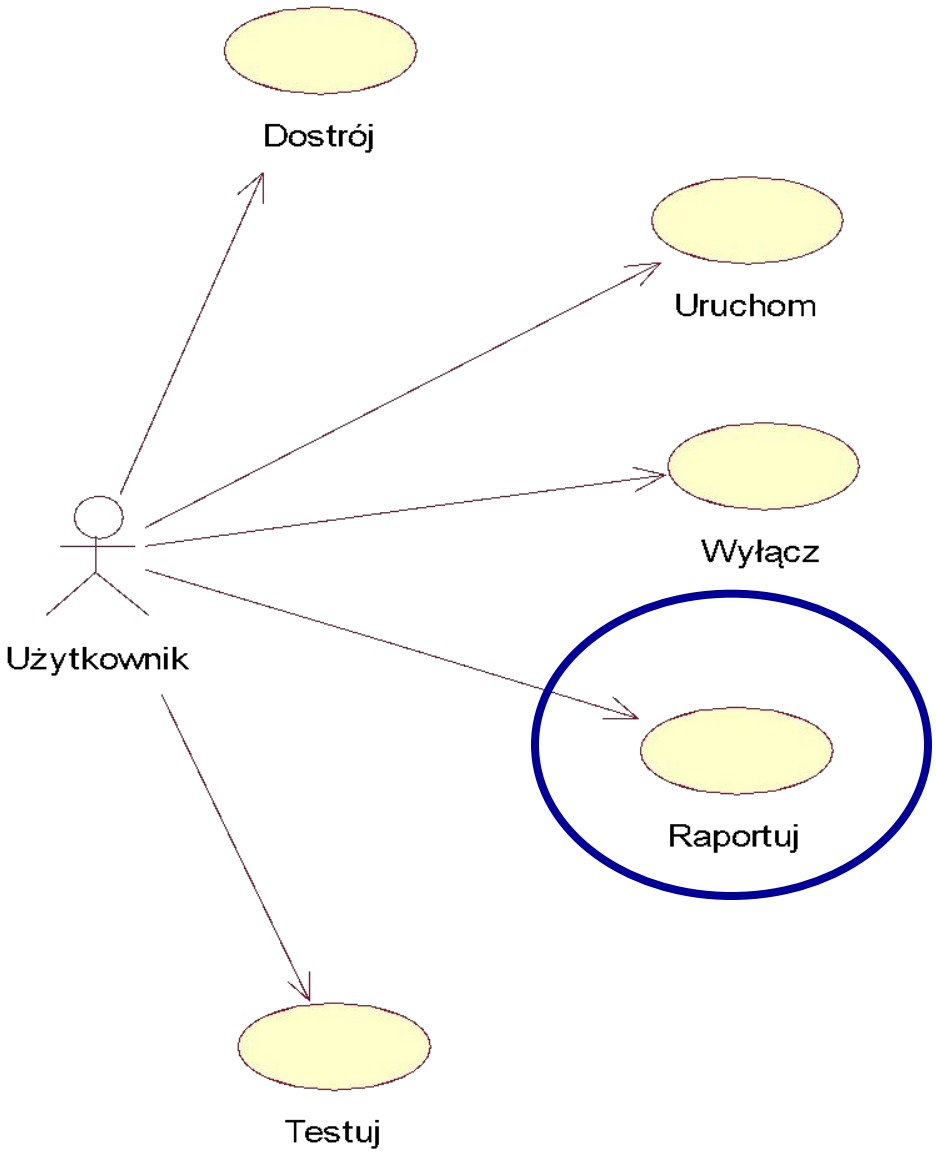
Podsystemy w systemie mapy pogody



Kontekst systemu i modele użycia systemu

- Pierwszym krok procesu projektowania oprogramowania:
  - zrozumienie związków między projektowanym oprogramowaniem a jego środowiskiem zewnętrznym;
  - określenie kontekstu systemu:
    - model statyczny;
    - tu jest to podsystem gromadzenia danych;
  - określenie modeli użycia systemu
    - model dynamiczny
    - opisuje, w jaki sposób system porozumiewa się ze swoim środowiskiem;

Przypadki użycia stacji meteorologicznej



Przypadki użycia stacji meteorologicznej

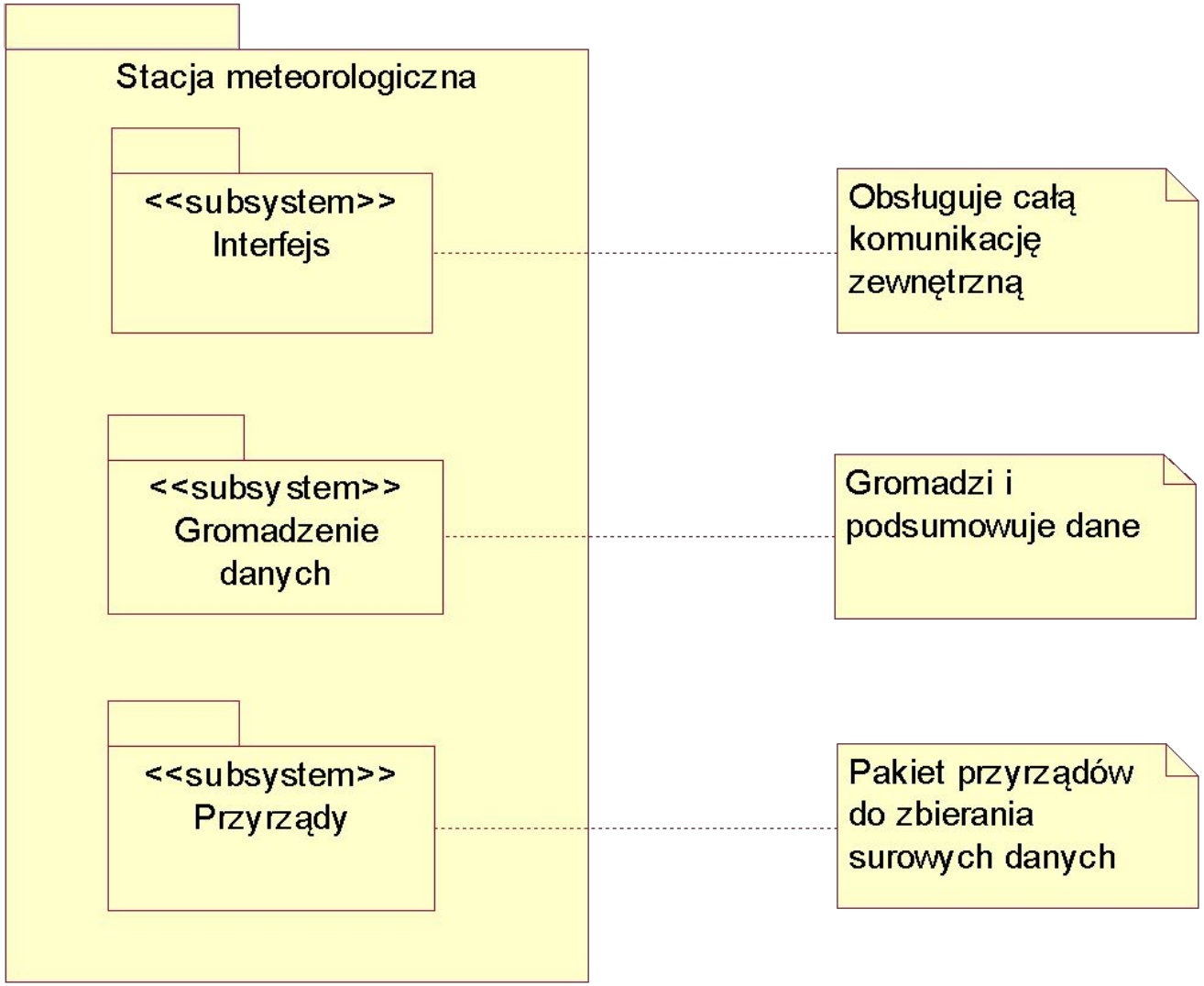
Opis przypadku użycia „Raportuj”

System	Stacja meteorologiczna
Przypadek użycia	Raportuj
Aktorzy	System gromadzenia informacji meteorologicznych, Stacja meteorologiczna
Dane	Stacja meteorologiczna wysyła do systemu gromadzenia informacji meteorologicznych podsumowanie danych meteorologicznych odczytanych z przyrządów w określonym czasie. Przesyłane dane to: maksymalne, minimalne i średnie temperatury gruntu i powietrza, maksymalne, minimalne i średnie ciśnienia powietrza, maksymalną, minimalną i średnią prędkość wiatru, całkowity opad i kierunek wiatru (co 5 minut).
Bodziec	System gromadzenia informacji meteorologicznych nawiązuje połączenie ze stacją meteorologiczną i wywołuje przekazanie danych.
Reakcja	Wysyłanie podsumowania danych do systemu gromadzenia informacji meteorologicznych.
Komentarz	Stacje meteorologiczne są proszone o raport zazwyczaj raz na godzinę. Ta częstotliwość może być inna dla różnych stacji i w przyszłości może ulec zmianie.

Projektowanie architektury

- Drugi krok procesu projektowania oprogramowania:
  - projektowanie architektury;
- Architektura na przykładzie automatycznej stacji meteorologicznej (model 3-warstwowy):
  - 1-warstwa interfejsu –
    - porozumiewanie się z innymi częściami systemu i oferowanie zewnętrznych interfejsów systemu;
  - 2-warstwa gromadzenia danych
    - zarządzanie odczytem danych z przyrządów i podsumowywanie danych meteorologicznych przed przesłaniem ich do systemu tworzącego mapy;
  - 3-warstwa przyrządów
    - pakiet przyrządów służących do gromadzenia surowych danych o warunkach pogodowych;

Architektura stacji meteorologicznej

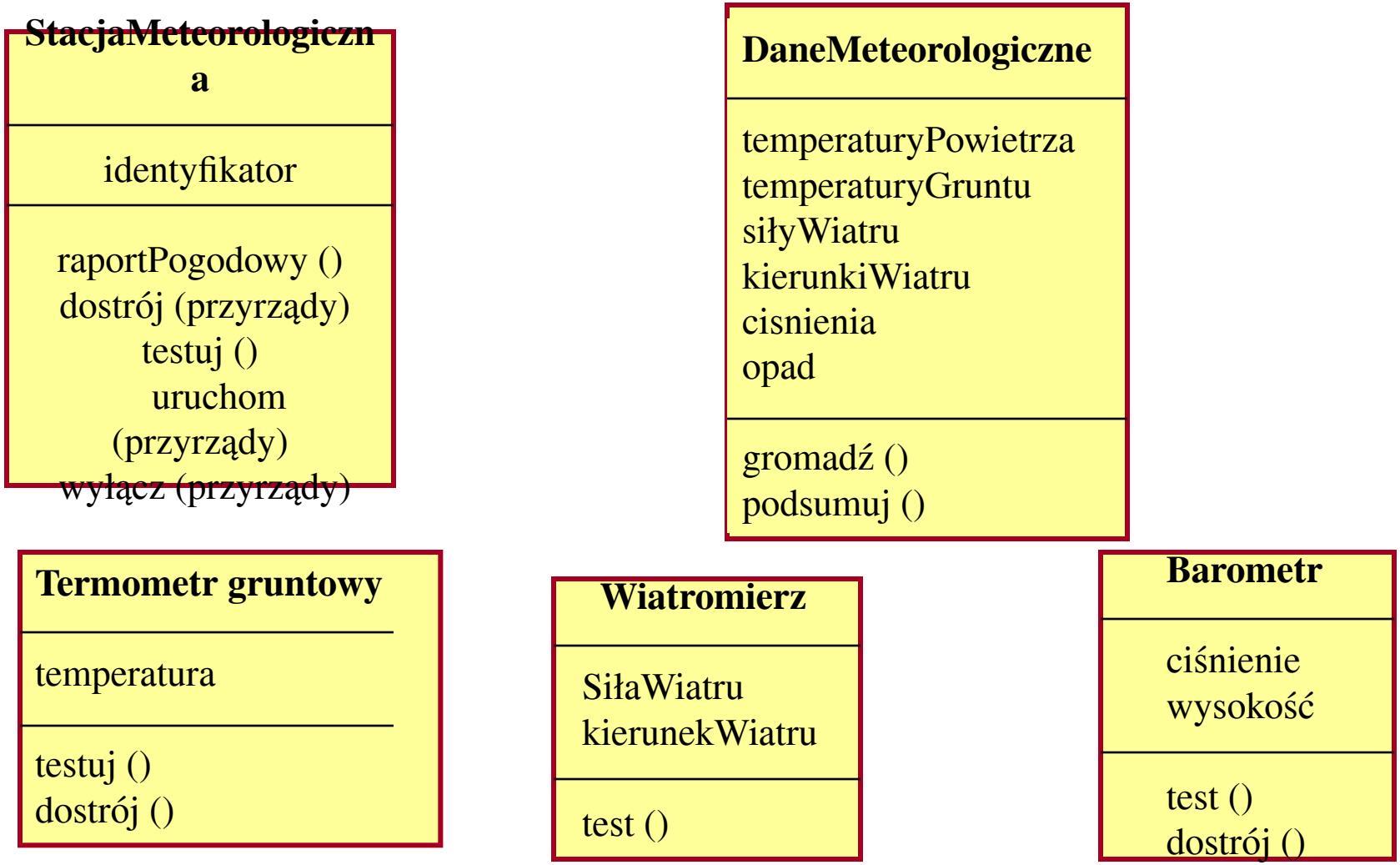


Klasy obiektów stacji meteorologicznej

- Trzeci krok procesu projektowania oprogramowania:
  - Identyfikacja (wynajdowanie) klas i obiektów;
- StacjaMeteorologiczna - oferuje podstawowy interfejs stacji meteorologicznej;
- DaneMeteorologiczne - jej operacje służą do gromadzenia i podsumowywania danych odczytanych z różnych przyrządów stacji meteorologicznej;
- Termometr gruntowy, Wiatromierz i Barometr - bezpośrednio związane z przyrządami systemu; odzwierciedlają namacalne byty sprzętowe systemu; operacje służą do sterowania tym sprzętem;

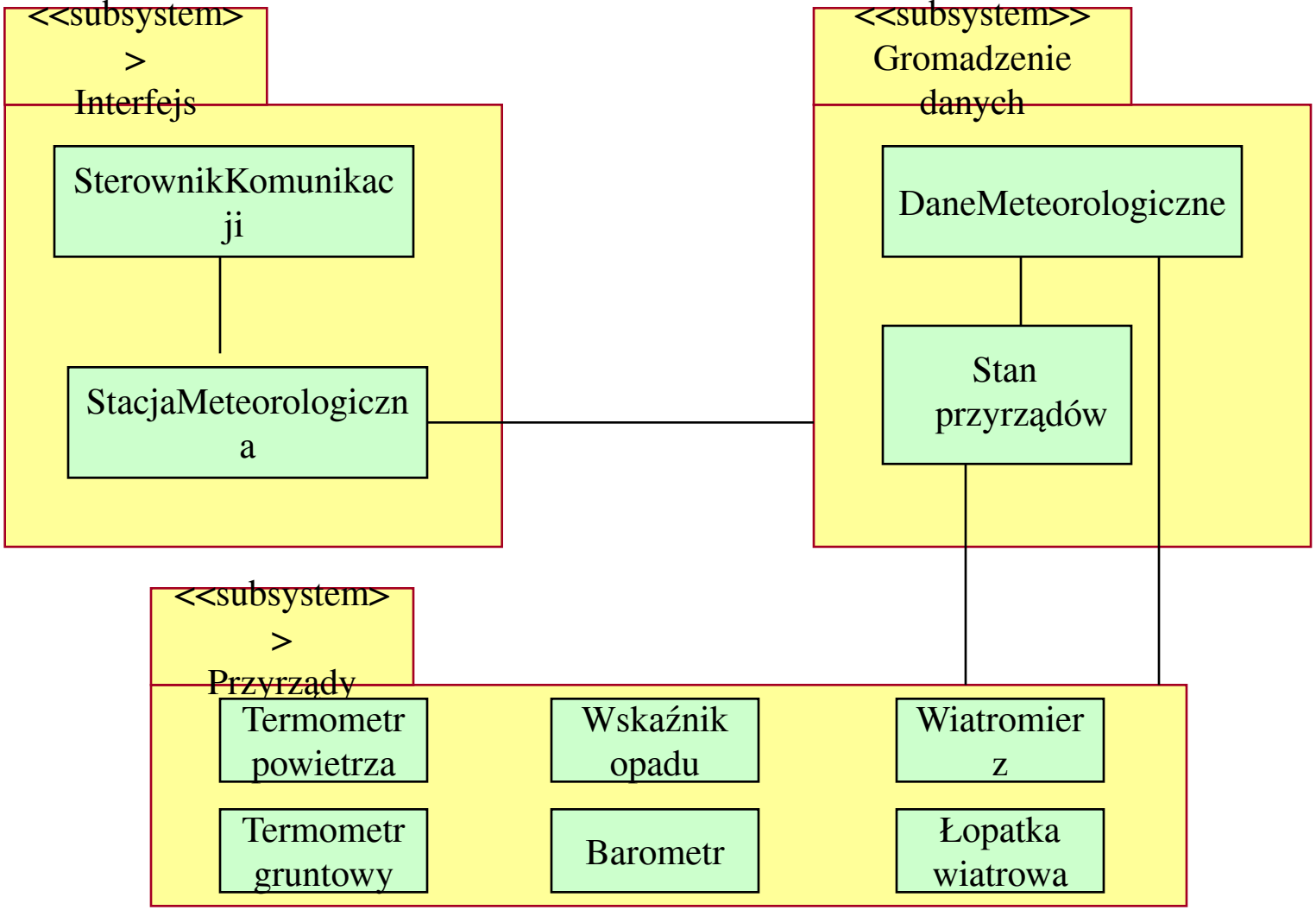
Klasy obiektów stacji meteorologicznej

Przykłady klas obiektów w systemie stacji meteorologicznej



Klasy obiektów stacji meteorologicznej

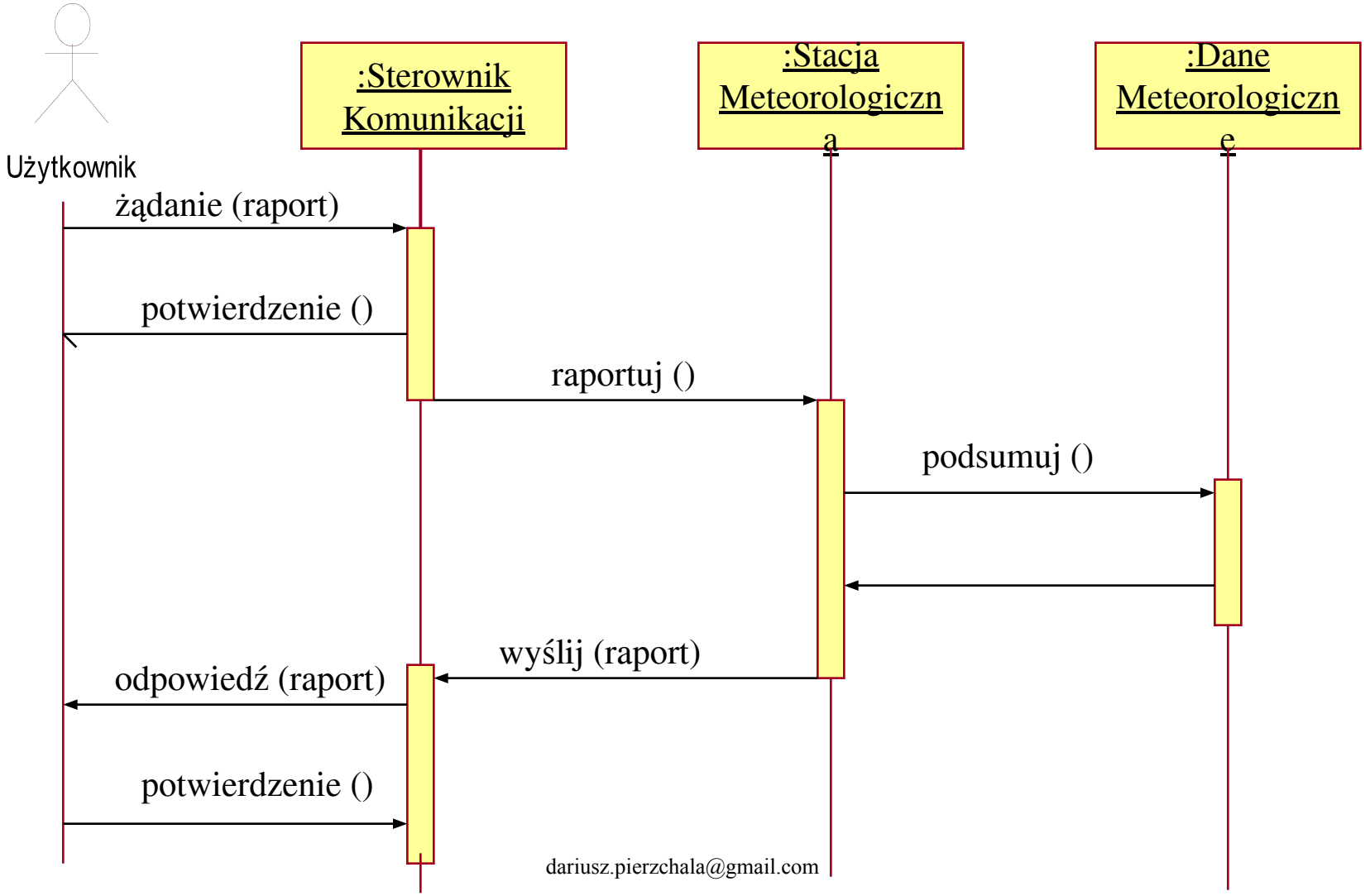
Przykład modelu podsystemów: powiązania obiektów stacji meteorologicznych



Sekwencja zdarzeń

- Czwarty krok procesu projektowania

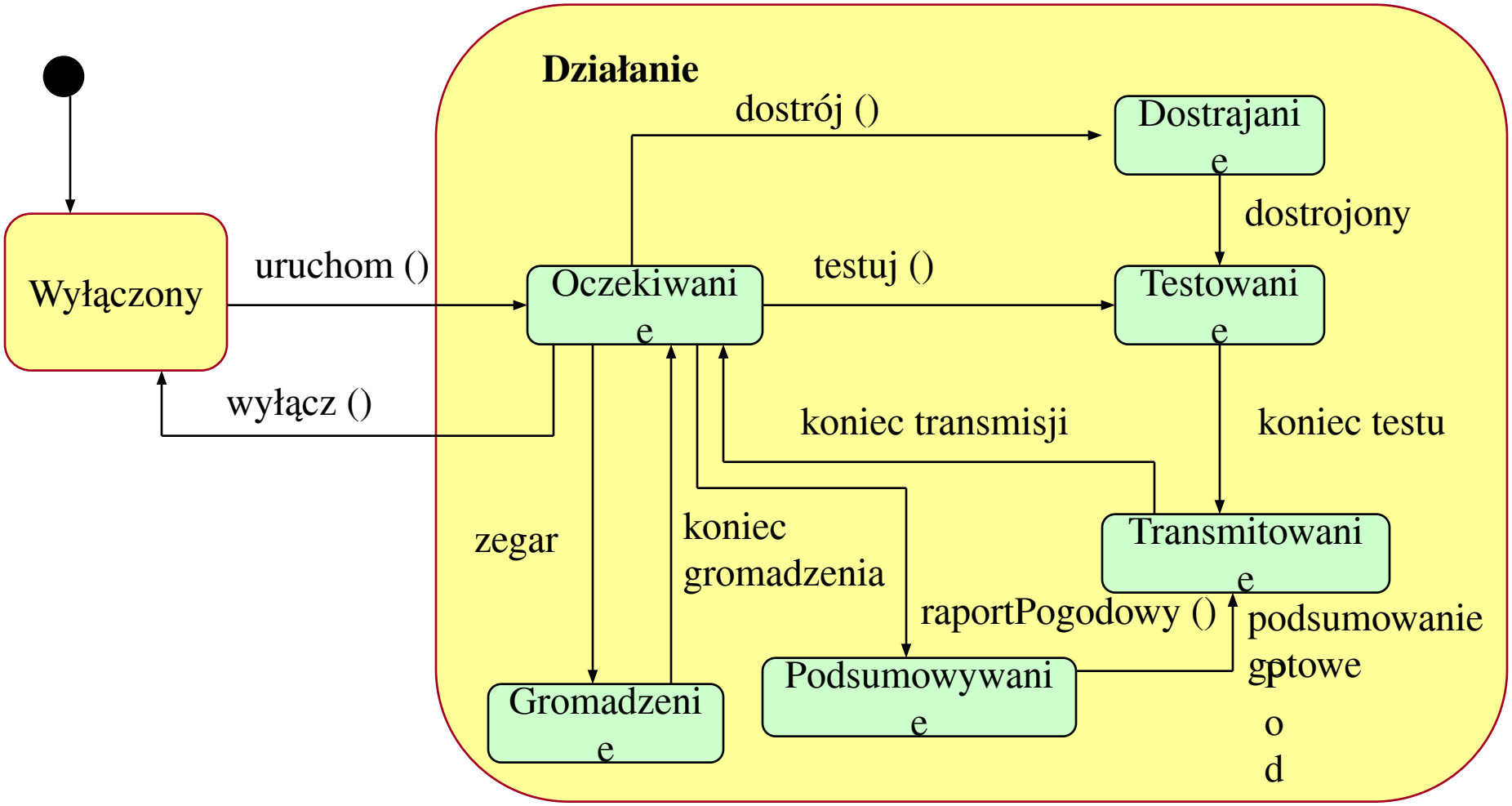
Przebieg operacji Gromadzenia danych



Diagramy stanów

- Piąty krok procesu projektowania

Przykład dla klasy StacjaMeteorologiczna



Specyfikowanie interfejsów obiektów

- Szósty krok procesu projektowania:
  - specyfikowanie interfejsów między komponentami;
- Pozwoli to na równoległe projektowanie komponentów;
- Jeden obiekt może mieć kilka interfejsów, które są sposobami postrzegania oferowanych metod;
- Realizacja w Java - interfejsy są deklarowane w oderwaniu od obiektów, a obiekty „implementują” interfejsy;

Specyfikowanie interfejsów obiektów

```
Interfejs StacjaMeteorologiczna {

   public StacjaMeteorologiczna () ;

   public void uruchom () ;
   public void uruchom (Przyrząd p) ;

   public void wyłącz () ;
   public void wyłącz (Przyrząd p) ;

   public void raportPogodowy () ;

   public void testuj () ;
   public void testuj (Przyrząd p) ;

   public void dostrój (Przyrząd p) ;

   public int podajID () ;

} // StacjaMeteorologiczna
```

Ewolucja projektu

- Kolejne kroki projektowania:
  - uszczegółowienie uproszczonego modelu;
- Zmiana wstępnie ustalonych szczegółów obiektu - nie wpłynie na inne obiekty systemowe;
- Wprowadzenie nowych obiektów - nie prowadzi do istotnych konsekwencji dla reszty systemu (obiekty są luźno powiązane);

Ewolucja projektu

- Do obiektu StacjaMeteorologiczna na tym samym poziomie co obiekt DaneMeteorologiczne należy dodać obiekt o nazwie Jakość powietrza;
- Należy dodać operację raport Jakości powietrza, której działanie polega na wysłaniu danych o zanieczyszczeniach do głównego komputera;
- Należy dodać obiekty reprezentujące przyrządy do pomiaru poziomu zanieczyszczeń. W tym wypadku pomiarom będą podlegać tlenek azotu, dym i benzen;

Ewolucja projektu

Nowe obiekty do monitorowania zanieczyszczeń